NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

IMPLEMENTATION OF REQUIREMENTS TRACING IN THE PROTOTYPING ENVIRONMENT UTILIZING PSDL

by

David J. Schmidt

December 1997

Thesis Advisor: Valdis Berzins

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
IMPLEMENTATION OF REQUIREMENTS TRACING IN THE PROTOTYPING ENVIRONMENT UTILIZING PSDL

5. FUNDING NUMBERS

6. AUTHOR(S) Schmidt, David J.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The prototyping description language (PSDL), a key component of CAPS, is a language designed for clarifying the requirements of complex real-time systems. Through the use of prototyping, the functional requirements for an embedded system can be rapidly validated to preclude inefficient usage of resources. This research has concentrated on the software engineering area of extending the PSDL data type and Ayacc source to support requirements tracing. Currently, CAPS doesn't use requirements tracing so the extensions just described are a significant step in that direction. This thesis includes an investigation into the potential use of an OODBMS which will interface with ADA95, and be utilized to store the list of requirement ids for each PSDL component.

Through the ADA95 program implementation and extension to the capabilities of the PSDL data type and Ayacc source, the programmer/designer has automated documentation support which will link the requirement ids to their respective component names. This research demonstrates there is no ADA95 OODBMS at the current time and therefore the requirement ids are stored in a file. There is an ADA95 OODBMS being developed at Lockheed Martin under the project name of FIRM. Also demonstrated is the connection of the unique list of requirement ids in the design phase with their respective PSDL components, so that the link between the design stages and analysis phase support for the modules is more completely established.

14. SUBJECT TERMS PSDL, OODBMS, ADA95, AYACC.			15. NUMBER OF PAGES 128
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

IMPLEMENTATION OF REQUIREMENTS TRACING IN THE PROTOTYPING ENVIRONMENT UTILIZING PSDL

David J. Schmidt
B.S., University of Nebraska, 1980
M.S., University of Nebraska, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1997

Author: David J. Schmidt
David J. Schmidt

Approved by: Valdis Berzins
Valdis Berzins, Thesis Advisor

Larry A. Palmer
Larry Palmer, Second Reader

Timothy J. Shimeall for Dan Boger
Dan Boger, Acting Chairman
Department of Computer Science

ABSTRACT

The prototyping description language (PSDL), a key component of CAPS, is a language designed for clarifying the requirements of complex real-time systems. Through the use of prototyping, the functional requirements for an embedded system can be rapidly validated to preclude inefficient usage of resources. This research has concentrated on the software engineering area of extending the PSDL data type and Ayacc source to support requirements tracing. Currently, CAPS doesn't use requirements tracing so the extensions just described are a significant step in that direction. This thesis includes an investigation into the potential use of an OODBMS which will interface with ADA95, and be utilized to store the list of requirement ids for each PSDL component.

Through the ADA95 program implementation and extension to the capabilities of the PSDL data type and Ayacc source, the programmer/designer has automated documentation support which will link the requirement ids to their respective component names. This research demonstrates there is no ADA95 OODBMS at the current time and therefore the requirement ids are stored in a file. There is an ADA95 OODBMS being developed at Lockheed Martin under the project name of FIRM. Also demonstrated is the connection of the unique list of requirement ids in the design phase with their respective PSDL components, so that the link between the design stages and analysis phase support for the modules is more completely established.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author would like to express his appreciation to Professor Valdis Berzins for his continued support for this thesis, in particular for his overall guidance in dealing with technical issues and questions that have arisen, and in routing specific support documentation without which forward progress would have been very difficult.

The author would also like to thank my second reader Larry Palmer, who was very helpful in editing and providing suggestions in the initial draft of this thesis.

I would also like to thank my parents who impressed upon me at an early age the importance of education. Finally, I would like to thank Catherine Good for her very important emotional support during this lengthy process of thesis development.

I. INTRODUCTION

A. GENERAL

This research concentrates on the software engineering area of extending the Prototyping System Description Language (PSDL) data type and Ada95 Yet Another Compiler-Compiler (Ayacc) source for requirements tracing. Currently, the Computer Aided Prototyping System (CAPS) does not include the requirements from the analysis phase. Each requirement is referred to by a unique id and so the extensions just described in the preceding abstract are a significant step in that direction. In addition, this thesis investigates the potential use of an Object Oriented Database Management System (OODBMS) which has an Application Programming Interface (API) for ADA95. Given that no OODBMS exists which interfaces with Ada95, a file is utilized to store the list of requirement ids and their respective PSDL component names. The stored information has recorded the association between requirement ids and the PSDL component names for those components that have requirement ids.

Through the extension to the capabilities of certain PSDL software packages and the Ayacc source file, "parser.y", the programmer/designer now has automated documentation support which links the requirement ids to their respective requirements and component names. By demonstrating the connection between the PSDL components and the associated list of requirement ids, the design and implementation phases are more completely supported by the analysis phase.

B. PROBLEM STATEMENT

As mentioned above, CAPS does not implement the requirement ids, however the PSDL grammar as described in Appendix A, "PSDL GRAMMAR", specifies the syntax to include the requirement ids and associate those ids with their respective components. A significant step in the direction of CAPS utilizing the requirement ids is to modify the PSDL data structure and the Ayacc source to support requirements tracing.

To store the requirement ids and other types of object data, it is necessary to investigate and implement, if possible, an OODBMS which interfaces to ADA95. A relational database cannot store all the various complex and abstract data objects utilized by the PSDL software. In addition, a relational database can be expensive with regard to performance since table joins are needed to follow pointers. The resources necessary to accomplish this task of locating an ADA95 OODBMS includes researching the library and World Wide Web (WWW) as well as utilizing email and the telephone.

C. SCOPE

This thesis concentrates on the extension of the capabilities of the PSDL data type and the Ayacc source to handle requirements tracing for PSDL modules. Investigation into an OODBMS which has an ADA95 API has also be conducted. Through modification of the Ayacc source file and the PSDL package software, it is possible to extract the PSDL component names and associated list of requirement ids. The requirement ids and associated components can be stored in an Object Oriented Database (OODB) or, if none exists, then in a file. The requirement ids refer to the individual requirements for the prototyping project used to create the functionality specified by the PSDL module.

The Ayacc source file has to be modified to capture the requirement ids for each PSDL component. The build component procedures in the Ayacc source file must include a parameter to hold the associated list of requirement ids. Selection of a data structure is needed to hold the requirement ids such that all ids including duplicate ones are stored. The make component procedures in the PSDL Ada software need to be modified to include a requirements variable. Furthermore, an output routine must be created to store the component names and associated list of requirements ids.

II. BACKGROUND

A. GENERAL

PSDL is a partially graphical text-based language utilized in real-time applications for specifying timing and functional behavior. The CAPS consists of an integrated set of software modules that support design, translation, and execution of prototypes. It is a software engineering tool designed to execute the specifications expressed in PSDL. The grammar for PSDL is completely specified in Appendix A. In addition, those PSDL Ada95 packages which either have been altered or created to handle the storage and retrieval of the requirements ids are located in Appendix G, "PSDL SOURCE CODE LISTING".

The Ayacc tool as described in Section C below is based on the popular Unix utility Yacc, and it closely imitates the features and conventions of its C counterpart [1]. The syntax, command line interface, and overall description of this utility is covered in Appendix B, "AYACC REFERENCE PAGES". Appendix H, 'AYACC SOURCE CODE LISTING", contains the modified Ayacc input file for the PSDL grammar.

B. PROTOTYPING SYSTEM DESCRIPTION LANGUAGE

The ever increasing demand for software systems is in the direction of larger, high quality systems which, up until now, have been very difficult, if not impossible, to create using current software development methods. To attain the goal of improved program productivity and reliability, a prototyping language such as PSDL is needed to support rapid prototyping [2]. A prototype is an executable version of a proposed software system which is used to guide the analysis and design work and can automatically generate the corresponding production code. Because the generated code very often does not fulfill all specifications of the proposed system, and in fact may not even operate within the targeted hardware configuration, it cannot be utilized as the final product [3].

PSDL is a language for describing prototypes and is utilized in real-time applications for specifying timing and functional behavior. It describes the proposed system as a hierarchy of networks consisting of processing units (operators) which communicate with instances of abstract data types by way of data streams [4]. PSDL has been used for the design of large embedded systems, feasibility studies, and requirements analysis. It is used to record and enforce timing constraints, as well as model control aspects relying on control constraints, operators, and data abstractions. The semantics of PSDL are not described, nor are those aspects related to hard real-time constraints and scheduling behavior, topics which are treated in [5].

The PSDL base is a computational model which, as mentioned above, contains Operators which communicate through Data Streams. Each stream can contain instances of a fixed abstract data type. The formal notation for the computational model is the augmented graph: G = (V, E, T(v), C(v)), where V is the set of vertices, E the set of edges, T(v) is the set of timing constraints for each vertex v, and C(v) is the set of control constraints for each vertex v. The vertices refer to operators while the edges are the data streams [2], and the first three components are the Enhanced Data Flow Diagram.

PSDL was designed to insure that prototypes be executable, simple, and easy to use; be based on a simple computational model encouraging good prototype modularization; support hierarchical structures for large-scale design; be applicable to both the specification and design levels, resulting in single uniform notation and avoidance of coding details; support multiple styles for formal and informal specifications; contain abstractions for constructing real-time systems; and, be able to specify the retrieval of reusable models from a software base [2] . The complete specification for the PSDL grammar is included in Appendix A.

1. Components

The two kinds of components in a PSDL prototype are operators and types. The firing of an operator can be described as at least reading one data object from each of the input streams and writing no more than one data object to each of its output streams. If

the operator is time critical in that there is at least one timing constraint associated with it, then there will be a maximum execution time (MET) in which other actions that can occur during firing will fit into this interval, including evaluating triggering conditions, calculating output values, and evaluating output guards [5]. Each operator is a state machine and those operators that have only one state are call functions [6]. It should be mentioned that the PSDL state machines cannot be shared by various subsystems to prevent hidden or unexpected interactions. Operators can either be atomic or composite. While the former cannot be decomposed in terms of the PSDL model, the latter can be decomposed as data and control flow networks of lower level operators [2].

The next component type refers to abstract data types the instances of which will be delivered by data streams (discussed in the next section). These types are immutable so there can be no communication involving side effects. Also worth noting are PSDL exceptions which are values of a built-in abstract data type called EXCEPTION. Exceptions are encoded just like data values to decouple the exceptions from the scheduling of actions for handling them [2].

Also important to note is that each PSDL component has both a specification and implementation part. The specification part has attributes describing the interface, timing characteristics, and formal and informal descriptions of the behavior of the operator. Those attributes consist of Generic Parameters, Input, Output, States, Exceptions and Timing Information [2]. The two kinds of implementations are the architecture descriptions and the code interface descriptions. The code interface descriptions define atomic components and specifies both the implementation language and the code module name, while the architecture descriptions have dataflow diagrams, control and timing constraints, requirement traces, and defines decomposition of composite components. In addition, the interface description determines if the operator is atomic or composite and for the architecture description there is a keyword specifying the underlying language followed by the name of the implementation module. The composite operators have the attributes Communication Graph, Internal Data, Control Constraints, and Informal Descriptions.

2. Data Streams

A stream is a communications link between one or more producer operators and one or more consumer operators. PSDL has two types of streams: a data flow and sampled stream. The data flow streams act like FIFO buffers which synchronize data-driven computations and whose values are not lost or replicated. Sampled streams act as continuously available data sources which can be read or updated on demand and whose values can be lost [4]. An example of a data flow stream would be the transactions in a system for electronic funds transfer. Also an example for sampled streams would be an operator that periodically updates a software estimate of the system state based on sensor readings [2].

3. Constraints

The control aspects of PSDL are specified implicitly and include whether an operator is sporadic or periodic, triggering conditions and output guards. A periodic operator has a period specified which controls its firing, while the sporadic operators are triggered by arrival of new data values possibly at irregular intervals. In addition, there are two types of data triggers for a PSDL operator designated as either 'Triggered by all' or 'Triggered by Some'. In the first case, the operator will fire when data has arrived on all the input streams. In the second case, the operator will fire when one/more of its inputs receives a new value [2]. A timer is a kind of state machine that behaves like a stopwatch and is used to affect such aspects of real time systems as time-outs or minimum refresh rates. The operations used for interacting with timers are Read, Start, Stop, and Reset. PSDL timers provide a non-local means of control [2]. Also in PSDL are two kinds of conditionals: the conditional execution of an operator and the conditional transmission of an output. The triggering condition is like a guard for the operator. If the predicate is satisfied, the operator will fire. Furthermore, the conditional transmission is an output guard which is a predicate that controls output of the operator.

Also worth mentioning are timing constraints. The most basic are those given in the PSDL specification part consisting of Maximum execution time (MET), Maximum response time(MRT) and the Minimum Calling Period(MCP). The MRT and MCP are important for sporadic operators. The MRT is an upper bound on the time between arrival of a new data value and the time when the last value is put into the output streams in response to the arrival of the new data value. The MCP consists of a lower bound on the delay between the arrival of a set of inputs and the arrival of the next set. Every sporadic operator with a MRT, must have a corresponding MCP [2]. Two important timing constraints for periodic operators are 'period' and 'finish within'. The former is the interval of time between consecutive triggering operator events and finish within is the upper bound on the time interval in which the MET occurs. Finally, the latency is the lower bound on the time in which data is fed into the stream and then read from it.

4. Conclusion

The PSDL language has been used for design of large embedded systems, feasibility studies, requirements analysis and the rapid prototyping of large real time systems. This section has introduced the reader to the important concepts (and their definitions) regarding the semantics of PSDL with minimal emphasis on hard real-time aspects of the language.

C. ADA95 YET ANOTHER COMPILER-COMPILER

The AYACC tool allows Ada programmers a means for automatic construction of parsers from a high level description of a context free grammar. The input to Ayacc consists of a BNF style specification of a grammar accompanied by a set of Ada program statements to be executed as each rule is recognized. Ayacc generates a set of Ada program units that acts as a parser for the specified grammar. These units can be interfaced to additional user supplied routines to produce a functional program [1]. As explained in Appendix B, this tool will produce the three packages: Tokens, Goto Table,

and Shift Reduce Table as well as a procedure called yyparse. The packages Goto Table and Shift Reduce Table contains the parsing tables used by yyparse, while Tokens contains the enumeration type returned by the lexical analyzer. It will be necessary for the user to supply yyparse with a lexical analyzer and an error reporting routine. The format and detailed explanation for the ayacc input file and the command line interface for Ayacc is also covered in the Appendix B.

The parser generated by Ayacc is like a finite state machine where the current state is at the top of the stack. During parsing, the parser uses its current state and the value of the next token obtained by calling a lexical analyzer such as those generated by the Ada Lexical Analyzer Generator (AFLEX) [7] to perform one of the following actions: ACCEPT, ERROR, SHIFT, or REDUCE defined in [1]. The aflex input file for PSDL, 'parser_lex.l', defines the lexical tokens for the ayacc PSDL parser. Aflex can either be used for simple lexical analysis and statistics, or with Ayacc as a parser front-end as is the case with the PSDL ayacc [7].

III. ADA95 OODBMS SEARCH ANALYSIS

A. GENERAL

This section describes the approaches and the results for the search analysis of an OODBMS which uses ADA95 as an API. As already mentioned, it is required that a variety of CAPS objects be stored in an OODBMS, and that an ADA95 programming language, a complete Object Oriented Programming (OOP) language, be utilized to access and manipulate those database objects. The following four sections delineate the four search methods and results: Library search method, World Wide Web search method, Email search method and finally the Phone search method. For each search method there is a corresponding appendix which lists the results for the particular method.

Briefly stated, an OODBMS includes a database that stores and allows retrieval of objects and classes. A real OODB provides complete support for objects, which means support for the following object and OOP features: encapsulation, inheritance, polymorphism, object identity and references among objects. Encapsulation is the process of combining data and code relevant to the data in one module in order to restrict access to the object. In the OODBMS, this feature refers to the ability to store code with data in the database [6]. Inheritance is the code-sharing mechanism by which new classes of objects can be defined in terms of an existing or base class. The classes derived from the base class can redefine the existing structure and behavior of the base class [6]. The database system must know the class hierarchy and manage the object storage. Polymorphism is defined as the ability to apply the same operation to different classes of objects. There are two ways this feature is implemented. One way is through subclassing where a defined method for a class is inherited by the subclasses. The second way is through overloading where the same name is used for methods in different classes [6]. When an object is read from the database, it has all the code and data members that it had when it was originally read. Object identity refers to the fact that each object in the OODB has a unique identity which distinguishes it from other objects [6]. For C++, the object address is the object identity, which allows pointer references to establish

relationships among objects. Object oriented database systems integrate the object identity in the database with the identity of objects in memory so that it is not necessary for the programmer to maintain this relationship between the two areas of storage. Object references are association links defined in each object's class so that objects can be made interdependent [6]. True object oriented database systems can automatically resolve pointer references in your program's objects and represents them in the database. In addition, some of the basic database operations that can be performed in an OODBMS include storing objects, changing objects, deleting objects and querying objects.

The Object Database Management Group (ODMG) is an organized group of object oriented database system vendors engaged in defining the standards for object oriented database systems. The initial standard, ODMG-93, published in September 1993, covers the essentials for object database programming especially the OO model and the OOP language bindings.

B. LIBRARY SEARCH RESULTS

The library search for an OODBMS with the Ada95 API was conducted at the Spawar Systems Center (SSC) topside library. The search was done utilizing the Knight-Ridder Information Inc., a world leader in electronic information access and delivery. This company offers the DIALOG and DataStar services which provide access to more than 600 online databases; KR OnDisc, a collection of nearly 70 files on CD-ROM; and KR SourceOne, a comprehensive, worldwide document delivery service. In Appendix C, "LIBRARY SEARCH LISTING, is the listing of papers that fulfill the library search criteria consisting of the following terms: Object Oriented Database System or OODBMS and Ada95 or Ada83. Ada83 was included in the search criteria to possibly increase the number of hits, even though this language is not object oriented but is object based. In the three papers cited, only the first two are of interest since the last is a repeat of the second paper listed.

The first paper I plan to comment on is listed in Appendix C, entitled "Ada/O2 Coupling: A Solution for an Efficient Management of Persistence in Ada 83", authored

by Thierry Millan and Pierre Bazex. This paper introduces the concept of persistence in Ada 83 through a data environment consisting of a set of variable identifiers, their linked data, and their types. Also worth mentioning from this paper is that a first prototype has been developed in Ada 83 which has not only shown the feasibility of the Ada/O2 coupling, but also the feasibility between the Ada data model and the ODMG - standardized object data model used by O2. This paper refers to a prototype but not an actual OODBMS which utilizes Ada 95. Also as mentioned below in Section E, "Phone Search Results", I've already contacted the O2 Technology company and they do not have a commercial OODBMS which has Ada 95 or Ada 83 as an API.

The second paper listed in Appendix C is entitled, "Ada persistence through an OODBMS O2", authored by Pierre Bazex, Thierry Milan and Frederic Mulatero. This paper discusses a method which allows for the introduction of persistence in Ada and how to represent a persistent environment in Ada 83/Ada 95 as well as the major problems encountered. As mentioned with the first paper, that paper refers to a prototype but not an actual OODBMS which utilizes Ada 95. In conclusion, the library search failed to locate a OODBMS which utilizes either Ada 83 or Ada 95 as an API.

C. WORLD WIDE WEB SEARCH RESULTS

In Appendix D, "WORLD WIDE WEB SEARCH LISTING", is a listing of WWW sites which I researched to locate an OODBMS that utilized Ada 83 or Ada 95 as an API. The initial site listed refers to the Ada Information Clearinghouse (AdaIC) which is sponsored by the Ada Joint Program Office and has been providing free information about Ada for over a decade. The AdaIC maintains close contact with the Ada community in order to obtain the latest information on a variety of Ada related topics to include tools and reusable code. I followed the relevant links without any success and also used their web search facility which again did not yield any success for an Ada API OODBMS.

The remaining sites investigated were accomplished utilizing the AltaVista search engine. AltaVista provides access to the largest Web index: 31 million pages found on

627,000 servers (1,158,000 host names), and four million articles from 14,000 Usenet news groups. It is accessed over 31 million times per weekday. The sets of search parameters were "ada95 and object oriented database management system and api" and "ada95 and oodbms". AltaVista retrieved 21 sites consisting of 22 links with 6 links being redundant so only the 16 unique links and their respective sites were included in Appendix D. Each of these unique sites were examined and following any further relevant links, I was unsuccessful in locating an OODBMS which utilizes Ada 95 as an API.

D. EMAIL SEARCH RESULTS

In Appendix E, "EMAIL SEARCH LISTING", is a listing of email contacts (and responses) which I initiated to locate an OODBMS that utilized Ada 83 or Ada 95 as an API. The first message was from "Michael Card", a technical representative for Lockheed Martin Corporation and a member of the ODMG. In this message Mr. Card discusses an OODBMS called the Functionally Integrated Resource Manager (FIRM) that is being developed under contract to Wright Laboratory. His paper [9] entitled, "FIRM: An Ada Binding to ODMB-93 1.2", details the Ada binding that the FIRM team designed using the ODMG object model and C++ language binding. As per the message, this product will not be available as a GOTS until sometime next year.

The second message included a response from Brad Balfour who is a member of ACM SIGAda and a developer at OIS System and formerly with CACI. According to Mr. Don Rosen of the AdaIC, Mr. Balfour is very knowledgeable about the Ada Community. From his message it is clear he is not aware of any commercial OODBMS products that offer an Ada API.

The third message listed in Appendix E where the response received was from Don Rosen the technical contact for the AdaIC. He was not aware of any Ada API tool that fits the requirements. He searched the AdaIC tools database for an OODBMS with an Ada95 API and posted an inquiry to the internet news group "comp.lang.ada" requesting

any commercial or shareware products that matched my requirements. In both cases the results were not positive.

The fourth message was from Douglas Barry, the president of Barry & Associates, Inc. and executive director of the ODMG. He was not aware of any commercial OODBMS with Ada95 as an API. His company maintains a web site at the following url: "http://www.odbmsfacts.com/choices.html/". This company provides facts on object databases, object-relational databases, object-mangers, and object-relational mapping products through consulting, publishing, and education. This url is listed in Appendix D. It was searched to locate the Ada API OODBMS without success.

E. PHONE SEARCH RESULTS

In Appendix F, "PHONE SEARCH LISTING", is a listing of database companies which I contacted to locate an OODBMS that utilized Ada 83 or Ada 95 as an API. This listing of companies was drawn from the last page of [10], and I listed the database companies in the Appendix from which I received replies regarding my OODBMS query. In [10], an overview was presented covering the OODBMS products and their features represented by various bar graph charts. I contacted a representative knowledgeable in there database management system product line for each company listed. In each instance I was told that their OODBMS did not have Ada95 as an API.

IV. REQUIREMENTS TRACING DESIGN

As mentioned previously the changes necessary to support requirements tracing involve extending the PSDL software and the Ayacc source file. The two main PSDL component types to be extended are the operator and the data type, each of which is specified in the PSDL grammar included in Appendix A. The Ayacc source file for the PSDL parser is included in Appendix H and consists of the following three sections: token declarations, rules and user declarations. The token declarations section is used to specify the generated tokens package. The rules section defines the grammar to be parsed. Each rule can include an associated action to be executed whenever the rule is recognized by the parser. The final user declarations section provides the 'with' and 'use' clauses for the tokens, parse table and text_io packages, calls the user supplied routines, YYLex and YYError, and can include other user defined functions or procedures.

Several global variables defined in user declarations section are involved in the building of the PSDL operator and type components. There are 12 global variables involved in the construction of the psdl type and 28 global variables utilized in the building of the psdl operator. Appendix H lists each of these variables and describes the corresponding data type for these types. The planned data type extension affects the initial builds defined in the action section for the data type and operator, which is included in the rules section of the parser source file. The actual builds occur in the package named psdl_component_pkg, which creates a record to completely define the data or operator types. The final design decision involves using the package, psdl_io, which provides the standard i/o facilities for the psdl program to output the data type or operator names and ids to a file. The psdl program source files affected by the above described changes are: psdl_io.ads, psdl_io.adb, psdl_component_pkb.ads, and psdl_component_pkb.adb. These packages are included in Appendix G.

V. REQUIREMENTS TRACING IMPLEMENTATION

A. GENERAL

The requirements variable, 'reqmts_trace' was utilized in the grammar section of the Ayacc source file. The implementation was included as far as defining the actions and procedures necessary to capture and store the ids per component. The definition of the requirements variable was similar to the keywords variable in that each consisted of a token followed by the variable, id_list. The id_list is defined as a one or more identifiers separated by commas and each identifier consists of a letter (includes underscore) followed by zero or more alphanumeric characters. Rather than define the requirements variable as the type 'psdl_id_set', the choice was the type 'psdl_id_sequence' because this type preserves all the ids whereas the former type only stores the unique ids. The same requirement id may occur in more than one place within a type or operator component, making it is important to use the correct type for storage purposes.

B. AYACC SOURCE EXTENSION

Aside from the way in which the requirements variable was defined as described above, the associated requirements global variable was created in the parser package body located in the final section of the Ayacc source file. Each time the requirements variable is parsed, the call is made to the psdl_id_sequence_pkg.append procedure which appends the current id sequence to the previous sequence (if any) and returns the new sequence of ids in the requirements variable. Each time a data type or operator is parsed, a call is made in the action segment to build the psdl type or operator respectively. The global requirements variable is added to the parameter list of the build calls since the requirement ids can be included in the operator or data type structures as described in the psdl grammar. The actual builds are performed in the package 'psdl_component_pkg', through procedure calls from within the final section of the Ayacc source file. After the build is performed, the 'output_trace' procedure is called which passes the component pointer. This procedure, as defined in the final section of the ayacc source file, first

checks that the global requirements variable is not empty before calling the 'put_reqm_ids' procedure, which is defined in the psdl_io package. The 'put_reqm_ids' procedure creates a file and stores each component name along with the respective set of requirement ids. After returning from the 'put_reqm_ids' procedure, the global requirements variable is reset to empty.

C. PSDL SOURCE EXTENSION

As described above, the actual builds occur in the package 'psdl_component_pkg'. Since the calls to the builds in the package are passing the global requirements variable, it was necessary to add another requirements variable in the appropriate places within each parameter list. Each operator or data type build consists of creating a record in which there exists a requirements id sequence field. Also a function called 'requirements' was created in this package which when called with a component pointer returns the sequence of requirements ids that were stored in the component. This function is similar to the function 'keywords' which when called with a component pointer returns the associated list of keywords. Therefore, another way to return the appropriate list of requirement ids to the 'output_trace' procedure involves calling this function rather than relying on the global requirements variable. Both approaches were utilized in various tests and in each case the same set of ids and component names were stored in a file.

The remaining procedure 'put_reqm_ids' as mentioned above stores the operator or data type name followed by the sequence of requirement ids in the file, 'test.reqs'. Each time this procedure is called, the file just mentioned is recreated, thereby wiping out old contents and storing only those data type or operator names that have requirement ids as well as the ids separated by commas.

VI. CONCLUSION

This research provides an important link between the analysis, design and implementation phases of a software engineering project by allowing the requirement ids to be captured and stored with their respective psdl components. With this vital link implemented, the analyst can be assured that the analysis phase more fully supports the later stages of the lifecycle in that all the requirements are represented in those phases, and that each step of the design process meets the necessary requirements.

This paper also demonstrated that currently there is no OODBMS with an ADA95 API. As mentioned previously, a Michael Card with Lockheed Martin is involved with a project developing an OODBMS called FIRM, and work on an ADA API could begin sometime next year. PSDL software relies on various complex data objects such as mappings, graphs and sequences and therefore requires an OODB for storage purposes [8]. Once an OODBMS with an ADA95 API becomes available, a recommended course of action would be to store the component names, requirement ids, and various PSDL abstract data types in such a database. Another suggestion would be to capture the segment information of each component that has requirements ids and store those variable names along with their ids. Through these suggestions, the analyst or designer can easily examine the database to extract all the necessary information regarding any PSDL program or PSDL component.

APPENDIX A. PSDL GRAMMAR

```
**************************************************************************

Optional items are enclosed in [ square brackets ]. Items which may
appear zero or more times appear in { braces }. Terminal symbols appear
in " double quotes ". Groupings appear in ( parentheses ).

**************************************************************************


psdl
     = {component}

component
     = data_type
     | operator

data_type
     = "type" id type_spec type_impl

type_spec
     = "specification" ["generic" type_decl] [type_decl]
       {"operator" op_name operator_spec}
       [functionality] "end"

operator
     = "operator" op_name operator_spec operator_impl

operator_spec
     = "specification" {interface} [functionality] "end"

interface
     = attribute [reqmts_trace]

attribute
     = "generic" type_decl
     | "input" type_decl
     | "output" type_decl
     | "states" type_decl "initially" initial_expression_list
     | "exceptions" id_list
     | "maximum execution time" time

type_decl
```

```
      = id_list ":" type_name {"," id_list ":" type_name}

type_name
      = id
      | id "[" type_decl "]"

id_list
      = id {"," id}

reqmts_trace
      = "required by" id_list

functionality
      = [keywords] [informal_desc] [formal_desc]

keywords
      = "keywords" id_list

informal_desc
      = "description" "{" text "}"

formal_desc
      = "axioms" "{" text "}"

type_impl
      = "implementation" id id "end"
      | "implementation" type_name {"operator" op_name operator_impl} "end"

operator_impl
      = "implementation" id id "end"
      | "implementation" psdl_impl "end"

psdl_impl
      = data_flow_diagram [streams] [timers] [control_constraints]
        [informal_desc]

data_flow_diagram
      = "graph" {vertex} {edge}

vertex
      = "vertex" op_id [":" time] {property}
        -- time is the maximum execution time

edge
      = "edge" id [":" time] op_id "->" op_id {property}
```

```
    -- time is the latency

property
    = "property" id "=" expression

op_id
    = [id "."] op_name ["(" [id_list] "|" [id_list] ")"]

streams
    = "data stream" type_decl

timers
    = "timer" id_list

control_constraints
    = "control constraints" constraint {constraint}

constraint
    = "operator" op_id
      ["triggered" [trigger] ["if" expression] [reqmts_trace]]
      ["period" time [reqmts_trace]]
      ["finish within" time [reqmts_trace]]
      ["minimum calling period" time [reqmts_trace]]
      ["maximum response time" time [reqmts_trace]]
      {constraint_options}

constraint_options
    = "output" id_list "if" expression [reqmts_trace]
    | "exception" id ["if" expression] [reqmts_trace]
    | timer_op id ["if" expression] [reqmts_trace]

trigger
    = "by all" id_list
    | "by some" id_list

timer_op
    = "reset timer"
    | "start timer"
    | "stop timer"

initial_expression_list
    = initial_expression {"," initial_expression}

initial_expression
    = "true"
```

```
        | "false"
        | integer_literal
        | real_literal
        | string_literal
        | id
        | type_name "." op_name ["(" initial_expression_list ")"]
        | "(" initial_expression ")"
        | initial_expression binary_op initial_expression
        | unary_op initial_expression

binary_op
        = "and" | "or" | "xor"
        | "<" | ">" | "=" | ">=" | "<=" | "/="
        | "+" | "-" | "&" | "*" | "/" | "mod" | "rem" | "**"

unary_op
        = "not" | "abs" | "-" | "+"

time
        = integer_literal unit

unit
        = "microsec"
        | "ms"
        | "sec"
        | "min"
        | "hours"

expression_list
        = expression {"," expression}

expression
        = "true"
        | "false"
        | integer_literal
        | time
        | real_literal
        | string_literal
        | id
        | type_name "." op_name ["(" expression_list ")"]
        | "(" expression ")"
        | expression binary_op expression
        | unary_op expression

op_name
```

```
    = id

id
    = letter {alpha_numeric}

real_literal
    = integer_literal "." integer_literal

integer_literal
    = digit {digit}

string_literal
    = """ {char} """

char
    = any printable character except "}"

digit
    = "0 .. 9"

letter
    = "a .. z"
    | "A .. Z"

alpha_numberic
    = letter
    | digit
    | "_"

text
    = {char}
```

APPENDIX B. AYACC REFERENCE PAGES

AYACC(local) MISC. REFERENCE MANUAL PAGES AYACC(local)

NAME

ayacc - An Ada LALR(1) parser generator

SYNOPSIS

ayacc {command line interface parameter associations}

DESCRIPTION

Ayacc provides Ada programmers with a convenient tool for the automatic construction of parsers from a high level description of a context free grammar. The input to Ayacc consists of a BNF-like specification of a grammar accompanied by a set of Ada program statements to be executed as each production is recognized. Ayacc outputs a set of Ada program units that act as a parser for the specified grammar; these program units may be interfaced to additional user-supplied routines to produce a functional program.

Ayacc will produce a procedure called yyparse and three packages: Tokens, Goto Table, and Shift Reduce Table. All of these packages must be visible to yyparse. Package Tokens contains the enumeration type that is returned by the lexical analyzer. Packages Goto table and Shift Reduce Table contain the parsing tables used by yyparse.

The user must supply yyparse with a lexical analyzer and an error reporting routine. The declarations of these routines should look like the following:

```
function YYLEX return TOKENS.TOKEN;

procedure YYERROR(MESSAGE: in STRING);
```

The format of the ayacc input file must be as follows,

```
declarations section
%%
rules section
%%
```

user declarations section

The declarations section is used to specify the generated tokens package. A token declaration consists of the keyword %token followed by a list of token names that may optionally be separated by commas. Token names must follow Ada enumeration type naming conventions. Ayacc provides a means to associate an Ada data type with a grammar symbol. This type must be called YYSType and must be declared in the tokens declarations section and be surrounded by '{' '}'s . e.g.


```
{
subtype YYSType is Integer;
}
```

Since types declared in this section may require visibility to additional packages, context clauses for the tokens package may be defined by using the keywords %with and %use. These keywords must be located before the YYStype declaration.

The rules section defines the grammar to be parsed. Each rules consists of a nonterminal symbol followed by a colon and a list of grammar symbols terminated by a semicolon. For example, a rule corresponding to a street address might be written as,

```
Address: Street City ',' State Zip;
```

A vertical bar may be used to combine rules with identical left hand sides. Nonterminal names may be made up of alphanumeric characters as well as periods and underscores. Ada reserved words are allowed. Unlike, yacc all tokens and nonterminal names are case insensitive. The start symbol of the grammar may be specified using the keyword %start followed by the symbol. If the start symbol is not specified, ayacc will use the left hand side of the first

grammar rule.

Ayacc allows each grammar rule to have associated actions which are executed whenever the rule is recognized by the parser. An action consists of Ada statements enclosed in braces and placed after the body of a rule. Ayacc uses a pseudo-variable notation to denote the values associate values with nonterminal and token symbols. The left hand side of a rule may be set to a value by an assignment to the variable, $$. For example, if YYSType is an integer, the action:

```
A : B C D {$$ := 1;}
```

sets the value of A to 1. Values of symbols on the right hand side of the rule, may be accessed through the variables $1..$n ,where n refers to the nth element of the right hand side. For example.

```
A : B '+' C {$$ := $1 + $3;}
```

sets A to the sum of the values of B and C.

The user declarations section is optional. By default, ayacc generates a parameterless procedure, YYParse. If the user desires, the procedure may be incorporated within a package provided in this section. The user must use the key marker, ##, to indicate where the body of YYParse is to be inserted. The user is responsible for providing with clauses for the tokens, parse table, and Text IO packages.

COMMAND LINE INTERFACE

Arguments are passed to ayacc via a command line interface. This command line interface models the syntax and semantics of Ada procedure calls, supporting both named and positional notation as well as default initialization. When the ayacc command is entered without arguments, the following specification is displayed.

```
-- Ayacc: An Ada Parser Generator.

type Switch is (On, Off);

procedure Ayacc (File      : in String;
                 C_Lex     : in Switch := Off;
                 Debug     : in Switch := Off;
                 Summary   : in Switch := On;
                 Verbose   : in Switch := Off;
                 Extension : in String := ".a");

-- File    Specifies the Ayacc Input Source File.
-- C_Lex   Specifies the Generation of a 'C' Lex
--         Interface.
```

-- DebugSpecifies the Production of Debugging Output	
--	By the Generated Parser.
-- Summary	Specifies the Printing of Statistics About the
--	Generated Parser.
-- Verbose	Specifies the Production of aHuman Readable
--	Report of States in the Generated Parser.
-- Extension	Specifies the File Extension to be Used for
--	Generated Ada Files.

The following examples are legal invocations of ayacc

ayacc ("file.y");

ayacc ("file.y, Verbose => On, Extension => ".ada"); Sun Release 4.1

Last change: 3 June 1988 3

AYACC(local) MISC. REFERENCE MANUAL PAGES AYACC(local)

For friendlier usage, some Ada rules have been relaxed,

1) Final semicolon on the procedure call is optional. 2) Outermost parentheses are optional.

3) Parentheses around aggregate parameters are optional when the aggregate consists of only one component.

4) Quotes around string literals are optional.

making the following examples legal invocations as well.

ayacc file.y
ayacc file.y Verbose => On Extenstion => .ada

Note: Unix will interpret the => and parantheses prior to passing arguments to alex. As a result, they must be escaped on the command line to prevent interpretation. i.e. => must be typed in as =\>.

FILES

file.y the input file to Ayacc	
file.ada the generated parser	
file.goto.ada	package Goto Table
file.shift_reduce.ada	package Shift Reduce Table
file.tokens.ada	package Tokens

file.verbose the verbose output

file.c_lex.ada package c lexforinterfacingwithlex
file.h the C include file for interfacing

with lex

BUGS
send questions and comment to ayacc-info@ics.uci.edu
send bug reports to ayacc-bugs@ics.uci.edu

SEE ALSO
Ayacc User's Manual
yacc(1), lex(1), alex(local) Sun Release 4.1 Last change: 3 June 1988
4

APPENDIX C. LIBRARY SEARCH LISTING

ds

Set Items Description

S1 3 (OBJECT()ORIENTED()DATABASE()MANAGEMENT()SYSTEM OR OODBMS AND (ADA95 OR ADA()95 OR ADA83 OR ADA()83)

S2 2 RD (unique items)

?t 1/5/all

1/5/1 (Item 1 from file: 2)

DIALOG(R)File 2:INSPEC

5398529 INSPEC Abstract Number: C9611-6160J-023

Title: Ada/O2 coupling: a solution for an efficient management of persistence in *Ada* *83*

Author(s): Millan, T.; Bazex, P.

Author Affiliation: IRIT, Univ. Paul Sabatier, Toulouse, France

Conference Title: Reliable Software Technologies - Ada-Europe '96. 1996 Ada-Europe International Conference on Reliable Software Technologies. Proceedings p.396-412

Editor(s): Strohmeier, A.

Publisher: Springer-Verlag, Berlin, Germany

Publication Date: 1996 Country of Publication: Germany xi+511 pp.

ISBN: 3 540 61317 X Material Identity Number: XX96-01477

Conference Title: Proceedings of Ada-Europe '96 Conference in cooperation with Reliable Software Technologies

Conference Date: 10-14 June 1996 Conference Location: Montreaux, Switzerland

Language: English Document Type: Conference Paper (PA) Treatment: Practical (P)

Abstract: We introduce the concept of persistence in *Ada* *83* through a data environment consisting of a set of variables' identifiers, their linked data and their types. In general this environment is represented as an oriented graph which expresses

the links between data. In that context, a persistent environment is a subset (subgraph) of a data environment defined from identifiers of persistent variables. The persistence of a data environment allows the subgraph to persist after the run-time of the program and so to be used again. We first describe the representation of a persistent environment in *Ada* *83* and its management through the O2 database management system. This system manages persistence, integrates the transactional aspect and manages simultaneous accesses. Finally, we present the problems inferred from persistence management through an *object*-*oriented* *database* *management* *system*. (18 Refs)

Descriptors: Ada; compiler generators; data structures; directed graphs; object-oriented databases; object-oriented programming; programming environments; transaction processing

Identifiers: *Ada* *83*; data environment; identifiers; linked data; types; oriented graph; persistent variables; subgraph; O2 *object*- *oriented* *database* *management* *system*; transactional aspect; simultaneous accesses

Class Codes: C6160J (Object-oriented databases); C6140D (High levellanguages); C6150C (Compilers, interpreters and other processors) Copyright 1996, IEE

1/5/2 (Item 2 from file: 2)

DIALOG(R)File 2:INSPEC

(c) 1997 Institution of Electrical Engineers. All rts. reserv.

5046395 INSPEC Abstract Number: C9510-6110J-039

Title: Ada persistence through an *OODBMS* O2

Author(s): Bazex, P.; Millan, T.; Mulatero, F.

Author Affiliation: IRIT, Univ. Paul Sabatier, Toulouse, France

Journal: Ada User Journal vol.16, no.2 p.71-82

Publication Date: June 1995 Country of Publication: Netherlands

CODEN: AUJOET ISSN: 0268-652X

Language: English Document Type: Journal Paper (JP)

Treatment: Practical (P)

Abstract: The paper presents a method that allows the introduction of persistence in Ada. An environment has a graph structure: is composed of a set of identifiers, their linked data and their types. A persistent environment is a subset of an environment, defined from identifiers called persistent roots, their types and all data linked to them. The persistence of an environment is a property that lets its subgraph remain existant when run-time ends. In order to be implicit, the persistence must respect the orthogonality, propagation and transparency rules. The paper shows how to represent a persistent environment in *Ada83* and *Ada95* and the major problems encountered. We are realizing a prototype in collaboration with O2-Technology. This company supplies the O2 software as the underlying system of our prototype. (17 Refs)

Descriptors: abstract data types; Ada; object-oriented databases; object-oriented languages; object-oriented programming; programming environments.

Identifiers: Ada persistence; O2 *OODBMS*; graph structure; identifiers; linked data; types; persistent environment; persistent roots; subgraph; orthogonality; propagation; transparency rules; *Ada83*; *Ada95*;
O2-Technology; O2 software

Class Codes: C6110J (Object-oriented programming); C6160J (Object-oriented databases); C6140D (High level languages); C6120 (File organisation); C6115 (Programming support)

1/5/3 (Item 1 from file: 8)

DIALOG(R)File 8:Ei Compendex(R)

04229402 E.I. No: EIP95082821464

Title: Ada persistence through an *OODBMS* O2

Author: Bazex, Pierre; Millan, Thierry; Mulatero, Frederic

Corporate Source: Universite Paul Sabatier, Toulouse, Fr

Source: Ada User Journal v 16 n 2 Jun 1995. p 71-82

Publication Year: 1995

CODEN: AUJOET ISSN: 0268-652X

Language: English

Document Type: JA; (Journal Article) Treatment: G; (General Review)

Journal Announcement: 9510W3

Abstract: This paper presents a method that allows the introduction of persistence in Ada. An environment has a graph structure: is composed of a set of identifiers, their linked data and their types. A persistent environment is a subset of an environment, defined from identifiers called persistent roots, their types and all data linked to them. The persistence of an environment is a property that lets its subgraph remain existent when run-time ends. In order to be implicit, the persistence must respect the orthogonality, propagation and transparency rules. This paper shows how to represent a persistent environment in *Ada83* and *Ada95* and the major problems encountered. We are realizing a prototype in collaboration with O2-Technology. This company supplies the O2 software as the underlying system of our prototype. (Author abstract)

Refs. Descriptors: *Ada (programming language); Database systems; Object oriented programming; Data structures; Graph theory; Computer software; Computer simulation

Identifiers: Ada persistence; Persistent environment; Persistent graph; Persistent root; Reusability

Classification Codes:

723.1.1 (Computer Programming Languages)

723.1 (Computer Programming); 723.3 (Database Systems); 723.2 (Data Processing);

921.4 (Combinatorial Mathematics, Includes Graph Theory, Set Theory);

723.5 (Computer Applications)

723 (Computer Software);

921 (Applied Mathematics)

72 (COMPUTERS & DATA PROCESSING);

92 (ENGINEERING MATHEMATICS) ?

APPENDIX D. WORLD WIDE WEB SEARCH LISTING

http://sw-eng.falls-church.va.us/AdaIC/tools

No Title

Newsgroups: comp.object,comp.answers,news.answers Path:...

http://puma.join.ad.jp/tech/faq-e/misc/misc-comp-object-faq - size 650K - 12 Apr 96

The Object-Oriented Page

The Object-Oriented Page. Last updated: December 05, 1996. Because OO info is increasingly populating the web and several excellent OO search engines and..

http://doradus.einet.net/galaxy/Engineering-and-Technology/Computer-Technology/Object-Oriented-Systems/ricardo-devis/oo.html - size 69K - 6 Dec 96

http://www.well.com/user/ritchie/oo.html - size 69K - 6 Dec 96

The Object-Oriented Page

The Object-Oriented Page. Last modified: September 25, 1995. You will find here a lot of links to object-oriented info, as well as comments on... http: //banjo.isse.kuis.kyoto-u.ac.jp/~subieta/NotMyPapers/ObOrLinks.html - size 44K - 29 Nov 95

Información sobre Tecnologías Orientadas a Objetos

Nuestro agradecimiento a. Ricardo Devis Botella Presidente de APTO2 Presidente de INFOPLUS S. L. por permitirnos usar su Página Orientada a Objetos. http: //www3.uniovi.es/UniOvi/Apartados/Otros/oviedo3/info3/princ.htm-size 56K-19 Jun 96

No Title

COMP.OBJECT FAQ Version: 1.0.8 Date: 5/31/1995 Author: Bob Hathaway Geodesic Systems,Inc. Cyberdyne Systems Corporation rjh@geodesic.com...http: //gedeon.ttt.bme.hu/~mosko/dbms/oodb.faq - size 617K - 27 Feb 96

Ada FAQ: Programming with Ada

From:M.Kempe@ieee.org(MagnusKempe)Newsgroups:comp.lang.ada,comp.answers,news.answers Distribution:world Subject: Ada FAQ: Programming with Ada (part. http://www.adahome.com/FAQ/programming.html - size 140K - 21 Oct 96

Flinders University, RAD

SEE Home | People | Publications | Projects | CS Dept Home] RAD: Research Afternoon Discussions. RAD is organised by the post graduates students, it is... http://tuvalu.cs.flinders.edu.au/Local/rad.html - size 12K - 25 Feb 97

No Title

techreport{SabatellaM1988a, author ="Sabatella, Marc", institution ="University of California Berkeley, Department of Computer Science", title="{B}arking. http://www.lpac.ac.uk/SEL-HPC/Articles/DataBase/comp.misc.bib - size 31K - 22 Jan 97

Persistence in Ada (25-Mar-1996)

Persistence in Ada. Persistence Extensions to Ada95. A joint project between Univ of Adelaide and DSTO to add support for (almost) orthogonal persistence..

http://www.dstc.edu.au/AU/staff/crawley/ada/persistence_index.html - size 2K-25Mar 96

PHOAKS: Resources for comp.object

People Helping One Another Know Stuff "Together, we know it all." More resources: Page 2. Page 3. Page 4. Top Posters. Volunteered Resources. Newsgroup...

http://weblab.research.att.com/phoaks/comp/object/resources0.html – size 63k – 27 Feb 97

Object Resource Lists

CORBA and Distributed Object Resources: Links to Related Topics

http://www.qds.com/people/apope/ap_resources.html - size 32K - 24 Jan 97

compilers & interpreters archive (miscellaneous)

compilers & interpreters archive. miscellaneous subject area. A BibTeX version of this database can be found here. Article links with a size in kbytes... http: //www.lpac.ac.uk/SEL-HPC/Articles/GeneratedHtml/comp.misc.html-size 32K-7 Mar 97

No Title

220 0 article. Path: news.itd.umich.edu!caen!uwm.edu!news.moneng.mei.com!bloom-beacon.mit.edu!senator-bedfellow.mit.edu!faqserv From: Bob Hathaway....

http://faq.sph.umich.edu/faq/files/object-faq/part11 - size 49K - 17 Jul 95

No Title

220 0 article. Path: news.itd.umich.edu!caen!hookup!bloom-beacon.mit.edu!senator-bedfellow.mit.edu!faqserv From: Bob Hathaway. Newsgroups:...

http://faq.sph.umich.edu/faq/files/object-faq/part2 - size 49K - 17 Jul 95

RIGmailer-archive: OOTS 96

OOTS 96. James W. Moore (moorej@mail04.mitre.org) Wed, 3 Jul 96 16:43:50 -0400.

Messages sorted by: [date][thread][subject][author]

Previous... http://pebbles.cs.utk.edu/mail-archive/0170.html - size 21K - 4 Jul 96

Barry & Associates, Inc.

Barry & Associates provides facts on object database, object-relational database, object-manager, and object-relational mapping products. http://www.odbmsfacts.com/ - size 5K - 17 Feb 97

APPENDIX E. EMAIL SEARCH LISTING

To answer your questions:

>1. Working with the Naval Postgraduate School in Monterey and we would like a public domain version of an OODBMS with Ada95 as the API?

Our ODBMS is being developed under contract to Wright Laboratory, which is part of the USAF Wright-Patterson AFB. The name of our ODBMS is FIRM, which is an acronym for Functionally Integrated Resource Manager. Our API is public (that PostScript doc at the AdaIC that I referred you to), and I am Lockheed Martin's representative to the ODMG. We are in the process of making Lockheed Martin a voting member of the ODMG and we plan to start an Ada working group next year to form an Ada binding chapter in ODMG. We expect that our Ada API will be the "starting point" for our work. FIRM is not yet completed, however. We demonstrated basic functionality last October, and this November we intend to demonstrate our database distribution capability, fault tolerance via dual-redundant ODBMS servers, and persistence (using Solaris file system) at our "CDR". We will also need to perform full-path testing of the ODBMS and build some supporting tools for it next year. When this testing has been completed, FIRM should be suitable for use. FIRM will be available as a GOTS product from Wright Laboratory.

>2. Is a version of your OODBMS available for demo or purchase, etc? Costs? No version of FIRM will be available prior to our CDR. After that, getting an "early-release" demo version would have to be negotiated with our management and Wright Laboratory.

>3. What platforms or other resources required to operate the OODBMS? The final target platform for the FIRM ODBMS has not been chosen. We are therefore building FIRM to be platform/OS independent. Our development environment is Sun SPARC/Solaris 2.5, but our target is intended to be a flight computer in an aircraft.

>4. client served? I'm not sure what you mean here. If you mean "client-server?" the answer is yes. The way we envision FIRM being used is to have "application methods" resident on the server. These "application methods" are Ada procedures and functions whose signatures have a CORBA IDL description. These functions and procedures would

be invoked by a CORBA call, and they would do the low-level stuff (find an object, iterate a collection, search an index etc). Thus, an IDL definition for "correlate two tracks" might be visible, and invoking this "application method" would result in Ada procedure(s) or function(s) being run on the server that would manipulate the required collections/relationships/ indices/atomic objects and return the desired result. As FIRM is intended for use in embedded real-time environments, its design is intended to minimize client-to-server messages.

>5. What features are/are not available with this OODBMS? What apis? This information is in the files I sent you. There is also an excellent overview of the FIRM program that was written by my colleague Mayford Roark, the author of the FIRM SRS. If you can get the files I sent you to print, I will send you the FIRM overview as well. If not, perhaps I can mail the overview to you as well. I hope this information is helpful. Please contact me if I can be of further assistance.

Michael P. Card

Lockheed Martin Ocean, Radar and Sensor Systems Division Syracuse

NY 13221 voice: (315)-456-3022

FAX: (315)-456-2414

e-mail:card@syr.lmco.com

SECOND MESSAGE:

At 12:57 PM 5/20/97 -0700, you wrote:

>>I received your name from the ADAIC and wanted to check with you on whether there exists an OODBMS with an ADA api? ...

We've actually been looking into this recently. Currently the answer is a qualified no. I haven't found any commercial OODBMS that offer an Ada API or access method. The only thing I have found is from a search of comp.lang.ada:

Subject: Ada Persistence Through an OODBMS O2

From: millan@caps.irit.fr (Thierry MILLAN)

Date: 1995/06/14 Message-Id: <3rmi1i$97i@irit.irit.fr>

Newsgroups: comp.databases.object,comp.lang.ada

Organization: IRIT-UPS, Toulouse France Keywords: Ada, OODBMS O2, Persistence
We have made a coupling between Ada and the OODBMS O2 as underlying system (Ada User Journal, UK, ...). The prototype allows to handle O2 and/or Ada persistent data in an Ada program. This handle is transparent for the ada user and allows to use the O2 graphic tools. For more details, you can join us (discussing, possible applications, ...) Thanks.
Thierry MILLAN & Frederic MULATERO
millan@irit.fr
<<<< I don't know if this is an in-house effort or what. You may also want to contact David Wheeler (wheeler@ida.org) as he was recently asking the same question on comp.lang.ada.
Brad Brad Balfour === Note new company, e-mail,& phone number ===
OIS, Inc. Brad.Balfour@ois.com 703/295-6533

THIRD MESSAGE:
Mr. Schmidt.
In response to your request for a database tool that stores Ada 95 objects, I performed searches in the areas listed below. I have not found a match yet, but am still awaiting responses from some of these sources and will notify you if I obtain more information. Please feel free to call or e-mail if you need additional information.
1. Searched the AdaIC's Ada 95 Tools Database. This database consists of all the tools/bindings we are aware of. It is a comprehensive list from industry, academia, and government. We obtained this information through searches on the World Wide Web, at industry events such as TRI-Ada , through press releases from companies announcing tool releases, reading relevant trade journals and magazines, and through follow-up to companies that design Ada tools. The database can be accessed on the World Wide Web at http://sw-eng.falls-church.va.us/AdaIC/tools

2. Posted an message to the internet news group "comp.lang.ada" asking the readers if they were aware of any commercial or shareware products that fit your requirements. Have not received any feedback yet, but we monitor this newsgroup daily.

FOURTH MESSAGE:

From: Douglas Barry

There is no commercial oodbms with ada95 as an api. I suggest you contact Michael Card at Lockheed Martin. His email is card@syr.lmco.com. Hope this helps.

Doug Barry

APPENDIX F. PHONE SEARCH LISTING

COMPANY NAME	PHONE
1. Fujitsu Software Corporation	408-432-1300
2. Gemstone	800-243-9369
3. Hewlett-Packard	800-637-7740
4. Informix	800-331-1763
5. Matisse Software	415-610-0367
6. O2 Technology	800-798-5454
7. Objectivity Inc.	303-331-1491
8. Oracle	800-542-1170
9. Osmos	714-380-6999
10. Persitence	415-342-3655
11. POET	415-286-4640
12. Versant	415-329-7571
13. Barry & Associates	612-892-6113
14. Object Design	415-286-6697
15. UniSQL Incorporated	800-451-3267

APPENDIX G. PSDL SOURCE CODE LISTING

```
------------------------------------------------------------------------
----
--   This package provides standard i/o facilities for psdl programs.
------------------------------------------------------------------------
----

  with parser, parser_tokens;
  with psdl_program_pkg; use psdl_program_pkg;
  with psdl_component_pkg; use psdl_component_pkg;
  with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
  with text_io; use text_io;
package psdl_io is

  REQS: File_Type;
  procedure get(item: in out psdl_program) renames parser.get;
    -- Raises syntax_error, semantic_error.

  procedure get(file: in file_type; item: in out psdl_program)
    renames parser.get;
    -- Raises syntax_error, semantic_error.

  procedure put_component_spec(c: in psdl_component);
    -- Raises psdl_component_pkg.undefined_component.

  procedure put_component_imp(c: in psdl_component; p: in psdl_program);
    -- Raises psdl_component_pkg.undefined_component.

  procedure put(p: in psdl_program);
    -- Raises psdl_component_pkg.undefined_component.

  procedure put(file: in file_type; p: in psdl_program);
    -- Raises psdl_component_pkg.undefined_component.

  procedure put_reqm_ids(pcomp: psdl_component; rqms: psdl_id_sequence);
    -- Output component name and associated ids only if any ids

  syntax_error: exception renames parser_tokens.syntax_error;
  semantic_error: exception renames parser.semantic_error;
end psdl_io;

with psdl_concrete_type_pkg ;
  use psdl_concrete_type_pkg ;
  with psdl_graph_pkg ;
  use psdl_graph_pkg ;
  with expression_pkg ;
  use expression_pkg ;


package body psdl_io is  use edge_set_pkg ;
  package tim_op_io is new enumeration_io ( timer_op_id    );
  package my_int_io is new integer_io ( integer    );
  use my_int_io ;
  package my_bool_io is new enumeration_io ( boolean    );
  use my_bool_io ;
  htab  : constant string  := "   " ;
  procedure put_component_name ( c  : in psdl_component   ) is
begin
```

```
if component_category ( c    )  = psdl_operator
then   put ( "OPERATOR "    );

else   put ( "TYPE "    );
  end if;
   put_line ( convert ( name ( c    )    )    );
   end put_component_name;
  procedure put_id_seq ( ids  : in psdl_id_sequence   ) is  i  : natural
:= 1  ;

begin
if not equal ( ids    ,
empty   )
then   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id) is
begin

if i  > 1
then  put ( ", "    );
   end if;
 put ( convert ( id    )    );
 i := i  + 1  ;
end generator_loop_body;
procedure execute_generator_loop is new psdl_id_sequence_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(ids  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
    end if;
   end put_id_seq;
  procedure put_id_set ( ids  : in psdl_id_set   ) is  i  : natural  :=
1  ;

begin
if not equal ( ids    ,
empty   )
then   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id) is
begin

if i  > 1
then  put ( ", "    );
   end if;
 put ( convert ( id    )    );
 i := i  + 1  ;
end generator_loop_body;
procedure execute_generator_loop is new psdl_id_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(ids  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
    end if;
   new_line;
```

```
   end put_id_set;
  procedure put_id_set ( ids  : in psdl_id_set  ;
 message  : in string  ) is  i  : natural  := 1  ;

begin
if not equal ( ids    ,
empty  )
then  put ( htab  & htab  & message  & " "    );
   put_id_set ( ids    );
    end if;
   end put_id_set;
  procedure put_smet ( o  : in operator   ) is
begin
if specified_maximum_execution_time ( o    )  /= undefined_time
then  put ( htab  & htab  & "MAXIMUM EXECUTION TIME"    );
   put_line ( integer ' image ( specified_maximum_execution_time ( o
)    )  & " MS"    );
    end if;
   end put_smet;
  procedure put_text ( t  : in text  ;
 message  : in string  ) is
begin
if not eq ( t    ,
empty  )
then  put ( htab  & htab  & message  & " "    );
   put_line ( convert ( t    )    );
    end if;
   end put_text;
  procedure put_type_name ( tname  : in type_name   ) is  i  : natural
:= 1  ;

begin  put ( convert ( tname . name    )    );

if not equal ( empty_type_declaration    ,
tname . gen_par  )
then  put ( ascii . l_bracket    );
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; tn: type_name) is
begin

if i  > 1
then  put ( ", "    );
   end if;
 put ( convert ( id    )  & ": "    );
 put_type_name ( tn    );
 i := i  + 1  ;
end generator_loop_body;
procedure execute_generator_loop is new type_declaration_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(tname . gen_par  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   put ( ascii . r_bracket    );
    end if;
   end put_type_name;
  procedure put_type_decl ( td  : in type_declaration  ;
 message  : in string := " "   ) is  i  : natural  := 1  ;
```

```
begin
if not equal ( empty_type_declaration    ,
td   )
then   put_line ( htab  & htab  & message    );
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; tn: type_name) is
begin

if i  > 1
then  put_line ( ","    );
   end if;
 put ( htab  & htab  & htab  & convert ( id    )  & ": "    );
 put_type_name ( tn    );
 i := i  + 1  ;
end generator_loop_body;
procedure execute_generator_loop is new type_declaration_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(td  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   new_line;
    end if;
   end put_type_decl;
  procedure put_expression ( e  : in expression   );
  procedure put_function_call ( e  : in function_call_expression   ) is
tname  : type_name  := adt_name ( e    )  ;
  first_time  : boolean  := true  ;

begin
if equal ( tname    ,
infix_type_name   )
then   put ( "("    );
   put_expression ( expression_sequence_pkg . fetch ( arguments ( e    )
,
1   )    );
   put ( " "    );
   put ( convert ( operation ( e    )    )    );
   put ( " "    );
   put_expression ( expression_sequence_pkg . fetch ( arguments ( e    )
,
2   )    );
   put ( ")"    );

elsif equal ( tname    ,
prefix_type_name   )   then   put ( convert ( operation ( e    )    )
);
   put ( "("    );
   put_expression ( expression_sequence_pkg . fetch ( arguments ( e    )
,
1   )    );
   put ( ")"    );

else   put_type_name ( adt_name ( e    )    );
   put ( "."    );
   put ( convert ( operation ( e    )    )    );
```

```
if expression_sequence_pkg . length ( arguments ( e    )    ) > 0
then   put ( "("    );
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(e: expression) is
begin

if first_time
then  first_time := false  ;

else  put ( ", "    );
 end if;
 put_expression ( e    );
end generator_loop_body;
procedure execute_generator_loop is new expression_sequence_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(arguments ( e    ) );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   put ( ")"    );
    end if;
  end if;
   end put_function_call;
  function int_to_string ( i  :  integer   ) return string is  s  :
constant string  := integer ' image ( i    )  ;

begin   return s ( 2  .. s ' last   )  ;
   end int_to_string;
  procedure put_expression ( e  : in expression   ) is  exp_cat  :
expression_category  := category ( e    )  ;

begin   case exp_cat   is  when undefined_exp   =>   put ( "?"    );
   when boolean_literal   =>
if boolean_value ( e    )
then   put ( "TRUE"    );

else   put ( "FALSE"    );
  end if;
  when integer_literal   =>   put ( int_to_string ( integer_value ( e
)    )    );
  when real_literal   =>   put ( convert ( text_of ( e    )    )    );
  when string_literal   =>   put ( convert ( text_value ( e    )    )
);
  when time_literal   =>   put ( int_to_string ( milliseconds ( e    )
)    );
   put ( " ms"    );
  when identifier   =>   put ( convert ( name ( e    )    )    );
  when function_call   =>   put_function_call ( e    );
  end case;
   end put_expression;
  procedure put_state ( state  : in type_declaration  ;
 init  : in init_map  ) is  initial_value  : expression  ;

begin
if not equal ( state    ,
empty_type_declaration   )
then   declare -- begin generator loop
return_from_generator_loop: exception;
```

```
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; tn: type_name) is
begin
 assign ( initial_value    ,
fetch ( init    ,
id   )   );
 put ( htab  & htab  & "STATES "  & convert ( id    )  & ": "    );
 put_type_name ( tn    );
 put ( " INITIALLY "    );
 put_expression ( initial_value    );
 new_line;
end generator_loop_body;
procedure execute_generator_loop is new type_declaration_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(state  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
    end if;
   end put_state;
  procedure put_operator_spec ( o  : in operator   ) is
begin   put_line ( htab  & "SPECIFICATION"    );
   put_type_decl ( generic_parameters ( o    )    ,
"GENERIC"   );
   put_type_decl ( inputs ( o    )    ,
"INPUT"   );
   put_type_decl ( outputs ( o    )    ,
"OUTPUT"   );
   put_state ( states ( o    )    ,
get_init_map ( o    )   );
   put_id_set ( exceptions ( o    )    ,
"EXCEPTIONS"   );
   put_smet ( o    );
   put_id_set ( keywords ( o    )    ,
"KEYWORDS"   );
   put_text ( informal_description ( o    )    ,
"DESCRIPTION"   );
   put_text ( axioms ( o    )    ,
"AXIOMS"   );
   put_line ( htab  & "END"    );
   end put_operator_spec;
  procedure put_op_spec_list ( op_map  : in operation_map   ) is
begin   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; o: operator) is
begin
 put ( htab    );
 put_component_name ( o    );
 put_operator_spec ( o    );
 new_line;
end generator_loop_body;
procedure execute_generator_loop is new operation_map_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(op_map  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   end put_op_spec_list;
```

```
  procedure put_type_spec ( t  : in data_type   ) is
begin   put_line ( "SPECIFICATION"    );
   put_type_decl ( generic_parameters ( t    )    ,
"GENERIC"   );
   put_type_decl ( model ( t    )    );
   put_op_spec_list ( operations ( t    )    );
   put_id_set ( keywords ( t    )    ,
"KEYWORDS"   );
   put_text ( informal_description ( t    )    ,
"DESCRIPTION"   );
   put_text ( axioms ( t    )    ,
"AXIOMS"   );
   put_line ( "END"    );
   new_line;
   end put_type_spec;
  procedure put_op_id ( o  : in op_id   ) is
begin
if not eq ( o . type_name    ,
empty   )
then   put ( convert ( o . type_name    )  & "."    );
    end if;
   put ( convert ( o . operation_name    )    );

if not ( equal ( o . inputs    ,
empty   )   and equal ( o . outputs    ,
empty   )    )
then   put ( "("    );
   put_id_seq ( o . inputs    );
   put ( "|"    );
   put_id_seq ( o . outputs    );
   put ( ")"    );
    end if;
   end put_op_id;
  procedure put_vertices ( g  : in psdl_graph   ) is  vertex_list  :
op_id_set  ;
  met  : millisec  ;

begin   assign ( vertex_list    ,
vertices ( g    )   );
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: op_id) is
begin
 put ( htab  & htab  & htab  & "VERTEX "    );
 put_op_id ( id    );
 met := maximum_execution_time ( id    ,
g   )  ;

if met  /= undefined_time
then  put_line ( ":"  & integer ' image ( met    )  & " MS"    );

else  new_line;
 end if;
 declare -- begin generator loop
exit_from_generator_loop: exception;
procedure generator_loop_body(p: psdl_id; e: expression) is
begin
 put ( htab  & htab  & htab  & htab  & "PROPERTY "  & convert ( p    )
& " = "    );
 put_expression ( e    );
```

```
 new_line;
end generator_loop_body;
procedure execute_generator_loop is new init_map_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(get_properties ( id    ,
g   )  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
end generator_loop_body;
procedure execute_generator_loop is new op_id_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(vertex_list  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   new_line;
   end put_vertices;
  procedure put_edges ( g  : in psdl_graph   ) is  edge_list  : edge_set
;
  latency_time  : millisec  ;

begin   assign ( edge_list    ,
edges ( g    )   );
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(e: edge) is
begin
 put ( htab  & htab  & htab  & "EDGE "  & convert ( e . stream_name    )
& " "    );
 latency_time := latency ( e . x    ,
e . y   ,
e . stream_name   ,
g   )  ;

if latency_time  /= undefined_time
then  put ( ":"  & integer ' image ( latency_time    )  & " MS "    );
   end if;
 put_op_id ( e . x    );
 put ( " -> "    );
 put_op_id ( e . y    );
 new_line;
 declare -- begin generator loop
exit_from_generator_loop: exception;
procedure generator_loop_body(p: psdl_id; e: expression) is
begin
 put ( htab  & htab  & htab  & htab  & "PROPERTY "  & convert ( p    )
& " = "    );
 put_expression ( e    );
 new_line;
end generator_loop_body;
procedure execute_generator_loop is new init_map_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(get_properties ( e . x    ,
e . y   ,
e . stream_name   ,
g   )  );
```

```
exception
when exit_from_generator_loop => null;
end; -- of generator loop
end generator_loop_body;
procedure execute_generator_loop is new edge_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(edge_list  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   new_line;
   end put_edges;
  procedure put_graph ( g  : in psdl_graph   ) is
begin   new_line;
   put_line ( htab  & htab  & "GRAPH"    );
   put_vertices ( g    );
   put_edges ( g    );
   end put_graph;
  procedure put_trigger ( oi  :  op_id  ;
 co  :  composite_operator  ) is  the_trigger  : trigger  := get_trigger
( oi    ,
co   )  ;
  the_exec_guard  : expression  ;

begin   assign ( the_exec_guard    ,
execution_guard ( oi    ,
co   )   );

if the_trigger . tt  = by_all
then   put ( htab  & htab  & htab  & htab  & "TRIGGERED BY ALL "    );
   put_id_set ( the_trigger . streams    );

elsif the_trigger . tt  = by_some  then   put ( htab  & htab  & htab  &
htab  & "TRIGGERED BY SOME "    );
   put_id_set ( the_trigger . streams    );
   end if;

if not eq ( the_exec_guard    ,
true_expression   )
then   put ( htab  & htab  & htab  & htab    );

if the_trigger . tt  = by_none
then   put ( "TRIGGERED "    );

else   put ( htab    );
  end if;
   put ( "IF "    );
   put_expression ( the_exec_guard    );
   new_line;
    end if;
  exception  when no_trigger  =>   null;
  end put_trigger;
  procedure put_timing ( time_val  :  millisec  ;
 timing_message  : in string  ) is
begin
if time_val  /= undefined_time
then   put ( htab  & htab  & htab  & htab  & timing_message    );
   put_line ( integer ' image ( time_val    )  & " MS"    );
    end if;
   end put_timing;
```

```
  procedure put_output_guard ( oi  :  op_id  ;
 parent  :  composite_operator ;
 p  :  psdl_program  ) is  local_oi  : op_id  := oi  ;
  local_parent  : composite_operator  := parent  ;
  op  : operator  ;
  out_streams  : type_declaration  ;
  og  : expression  ;

begin   op := get_definition ( p    ,
oi   )  ;

if op  = null_component
then   set_output ( standard_error    );
   put ( standard_error    ,
"-- undefined operator: "   );
   put_op_id ( oi    );
   new_line;
   set_output ( standard_output    );
   raise undefined_component;
    end if;
   out_streams := outputs ( op    )  ;
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; tn: type_name) is
begin
 assign ( og    ,
output_guard ( local_oi    ,
id   ,
local_parent   )   );

if not eq ( og    ,
true_expression   )
then  put ( htab  & htab  & htab  & htab    );
 put ( "OUTPUT "  & convert ( id    )  & " IF "    );
 put_expression ( og    );
 new_line;
   end if;
end generator_loop_body;
procedure execute_generator_loop is new type_declaration_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(out_streams  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   end put_output_guard;
  procedure put_timer_op ( oi  :  op_id  ;
 co  :  composite_operator  ) is  the_timer_op_list  : timer_op_set  ;

begin   the_timer_op_list := timer_operations ( oi    ,
co   )  ;
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(the_timer_op_rec: timer_op) is
begin
 put ( htab  & htab  & htab  & htab    );
 tim_op_io . put ( the_timer_op_rec . op_id    );
 put ( " TIMER "    );
 put ( convert ( the_timer_op_rec . timer_id    )    );
```

```
if eq ( the_timer_op_rec . guard    ,
true_expression   )
then  new_line;

else  put ( " IF "    );
 put_expression ( the_timer_op_rec . guard    );
 new_line;
 end if;
end generator_loop_body;
procedure execute_generator_loop is new timer_op_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(the_timer_op_list  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   end put_timer_op;
  procedure put_excep_trigger ( oi   :  op_id  ;
 parent  : in composite_operator ;
 p  : in psdl_program  ) is  local_oi  : op_id  := oi  ;
  local_parent  : composite_operator  := parent  ;
  op  : operator  ;
  the_exception_set  : psdl_id_set  ;
  et  : expression  ;

begin   op := get_definition ( p    ,
oi   )  ;

if op  = null_component
then   set_output ( standard_error    );
   put ( standard_error    ,
"-- undefined operator: "   );
   put_op_id ( oi    );
   new_line;
   set_output ( standard_output    );
   raise undefined_component;
    end if;
   the_exception_set := exceptions ( op    )  ;
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(e_name: psdl_id) is
begin
 assign ( et    ,
exception_trigger ( local_oi    ,
e_name   ,
local_parent   )   );

if eq ( et    ,
true_expression   )
then  put ( htab  & htab  & htab  & htab    );
 put_line ( "EXCEPTION "  & convert ( e_name    )    );

elsif not eq ( et    ,
false_expression   )  then  put ( htab  & htab  & htab  & htab    );
 put ( "EXCEPTION "  & convert ( e_name    )  & " IF "    );
 put_expression ( et    );
 new_line;
  end if;
end generator_loop_body;
```

```
procedure execute_generator_loop is new psdl_id_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(the_exception_set  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   end put_excep_trigger;
  procedure put_control_constraint ( oi   :  op_id  ;
 co  : in composite_operator ;
 p  : in psdl_program  ) is
begin   put ( htab  & htab  & htab  & "OPERATOR "    );
   put_op_id ( oi    );
   new_line;
   put_trigger ( oi    ,
co   );
   put_timing ( period ( oi    ,
co   )    ,
"PERIOD"   );
   put_timing ( finish_within ( oi    ,
co   )    ,
"FINISH WITHIN"   );
   put_timing ( minimum_calling_period ( oi    ,
co   )    ,
"MINIMUM CALLING PERIOD"   );
   put_timing ( maximum_response_time ( oi    ,
co   )    ,
"MAXIMUM RESPONSE TIME"   );
   put_output_guard ( oi    ,
co   ,
p   );
   put_timer_op ( oi    ,
co   );
   put_excep_trigger ( oi    ,
co   ,
p   );
   end put_control_constraint;
  procedure put_control_constraints ( co  : in composite_operator  ;
 p  : in psdl_program  ) is  the_op_id_set  : op_id_set  := empty  ;
  local_co  : composite_operator  := co  ;
  local_p  : psdl_program  := p  ;

begin   assign ( the_op_id_set    ,
vertices ( graph ( co    )    )   );
   put_line ( htab  & htab  & "CONTROL CONSTRAINTS"    );
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: op_id) is
begin
 put_control_constraint ( id    ,
local_co   ,
local_p   );
end generator_loop_body;
procedure execute_generator_loop is new op_id_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(the_op_id_set  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
```

```
   end put_control_constraints;
  procedure put_operator_implementation ( o  : in operator  ;
 p  : in psdl_program  ) is  co  : composite_operator  ;

begin   new_line;
   put ( htab  & "IMPLEMENTATION "    );

if component_granularity ( o    )  = composite
then   put_graph ( graph ( o    )    );
   put_type_decl ( streams ( o    )    ,
"DATA STREAM"   );
   put_id_set ( timers ( o    )    ,
"TIMER"   );
   put_control_constraints ( o    ,
p   );
   put_text ( implementation_description ( o    )    ,
"DESCRIPTION"   );

else   put_line ( "ADA "  & convert ( ada_name ( o    )    )    );
  end if;
   put_line ( htab  & "END"    );
   new_line;
   end put_operator_implementation;
  procedure put_type_implementation ( t  : in data_type  ;
 p  : in psdl_program  ) is  o  : operator  ;

begin   put ( "IMPLEMENTATION "    );

if component_granularity ( t    )  = composite
then   put_type_name ( data_structure ( t    )    );
   new_line;
   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; op: operator) is
begin
 put_line ( htab  & "OPERATOR "  & convert ( id    )    );
 put_operator_implementation ( op    ,
p   );
 new_line;
end generator_loop_body;
procedure execute_generator_loop is new operation_map_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(operations ( t    )  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop

else   put_line ( " ADA "  & convert ( ada_name ( t    )    )    );
  end if;
   put_line ( "END"    );
   end put_type_implementation;
  procedure put_component_spec ( c  : in psdl_component   ) is
begin   put_component_name ( c    );

if component_category ( c    )  = psdl_operator
then   put_operator_spec ( c    );

else   put_type_spec ( c    );
  end if;
```

```
   end put_component_spec;
  procedure put_component_imp ( c  : in psdl_component  ;
 p  : in psdl_program  ) is
begin
if component_category ( c    ) = psdl_operator
then   put_operator_implementation ( c    ,
p   );

else   put_type_implementation ( c    ,
p   );
  end if;
   end put_component_imp;
  procedure put ( p  : in psdl_program   ) is
begin   declare -- begin generator loop
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(id: psdl_id; c: psdl_component) is
begin
 put_component_name ( c    );

if component_category ( c    ) = psdl_operator
then  put_operator_spec ( c    );
 put_operator_implementation ( c    ,
p   );

else  put_type_spec ( c    );
 put_type_implementation ( c    ,
p   );
 end if;
end generator_loop_body;
procedure execute_generator_loop is new psdl_program_map_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(p  );
exception
when exit_from_generator_loop => null;
end; -- of generator loop
   end put;
  procedure put ( file  : in file_type  ;
 p  : in psdl_program  ) is
begin   set_output ( file    );
   put ( p    );
   set_output ( standard_output    );
   end put;
procedure put_reqm_ids(pcomp: psdl_component; rqms: psdl_id_sequence) is
  i: natural := 1;
begin
    if not Is_Open(REQS) then
      create(REQS,Out_File,"test.reqs");
    end if;
    put_line(REQS," ");
    if component_category(pcomp) = psdl_operator
      then put(REQS,"OPERATOR ");
      else put(REQS,"TYPE ");
    end if;
    put_line(REQS,convert(name(pcomp)));
    declare  -- begin generator loop
    return_from_generator_loop: exception;
    exit_from_generator_loop: exception;
    procedure generator_loop_body(id: psdl_id) is
    begin
```

```
    if i > 1 then
      put(REQS,", ");
    end if;
    put(REQS,convert(id));
    i := i + 1;
    end generator_loop_body;
    procedure execute_generator_loop is new
psdl_id_sequence_pkg.scan(generator_loop_body);
    begin
    execute_generator_loop(rqms);
    exception
     when exit_from_generator_loop => null;
    end;  -- of generator loop

end put_reqm_ids;
   end psdl_io;


-------------------------------------------------------------------------
----
-- specification for the psdl adt
-------------------------------------------------------------------------
----

  with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
  with psdl_graph_pkg; use psdl_graph_pkg;
  with expression_pkg; use expression_pkg;
  with generic_map_pkg;
package psdl_component_pkg is
-- by requirements clauses are ignored in this version.

-- discriminant types and associated declarations.
  type component_type is(psdl_operator, psdl_type);
  type implementation_type is(atomic, composite);

  type psdl_component_record
    (category: component_type; granularity: implementation_type)
  is private;

-- Main types
  type psdl_component is access psdl_component_record;
  subtype operator is psdl_component;  -- (category => psdl_operator).
  subtype data_type is psdl_component;  -- (category => psdl_type).
  subtype atomic_component is psdl_component;  -- (granularity =>
atomic).
  subtype atomic_operator is
    operator(category => psdl_operator, granularity => atomic);
  subtype composite_operator is
    operator(category => psdl_operator, granularity => composite);
  subtype atomic_type is
    data_type(category => psdl_type, granularity => atomic);
  subtype composite_type is
    data_type(category => psdl_type, granularity => composite);

  null_component: constant psdl_component := null;
    -- The default value for an operation map.
  function eq(x, y: psdl_component) return boolean;
  -- function eq(x, y: operator) return boolean;

  package operation_map_pkg is
          new generic_map_pkg(key => psdl_id, result => operator,
```

```
                                eq_key => eq, eq_res => eq,
                                average_size => 10);
   subtype operation_map is operation_map_pkg.map;
   -- An operation map is an environment that binds
   -- psdl operator names to psdl operator definitions.

   function empty_operation_map return operation_map;
   procedure bind_operation(key: in psdl_id; result: in operator;
                            map: in out operation_map)
     renames operation_map_pkg.bind;
   function fetch(m: operation_map; x: psdl_id) return operator
     renames operation_map_pkg.fetch;

   -- exception declarations
   undefined_component: exception;
   initial_state_undefined: exception;
   no_data_structure: exception;
   no_trigger: exception;
   input_redeclared: exception;
   output_redeclared: exception;
   state_redeclared: exception;
   initial_value_redeclared: exception;
   exception_redeclared: exception;
   specified_met_redefined: exception;
   not_a_subcomponent: exception;
   period_redefined: exception;
   finish_within_redefined: exception;
   minimum_calling_period_redefined: exception;
   maximum_response_time_redefined: exception;

   -- The following exceptions signal failures of explicit runtime
   -- checks for violations of subtype constraints.
   -- This is needed because ada does not allow partially constrained
types:
   -- if any discriminants are constrained, then all must be constrained.
   not_an_operator: exception;
   -- raised by operations on psdl operators that have an actual
parameter
   -- of type operator with category = psdl_type.
   not_a_type: exception;
   -- raised by operations on psdl data types that have an actual
parameter
   -- of type data_type with category = psdl_operator.
   not_an_atomic_component: exception;
   -- raised by operations on atomic components that have an actual
parameter
   -- of type atomic_component with granularity = composite.
   not_a_composite_component: exception;
   -- raised by operations on atomic components that have an actual
parameter
   -- of type atomic_component with granularity = composite.

   -- operations on all psdl components

   function component_category(c: psdl_component) return component_type;
   -- Indicates whether c is an operator or a type.

   function component_granularity(c: psdl_component) return
implementation_type;
   -- Indicates whether c is atomic or composite.
```

```
  function name(c: psdl_component) return psdl_id;
  -- Returns the psdl name of the component.

  function generic_parameters(c: psdl_component) return
type_declaration;
  -- Returns an empty type_declaration if no generic parameters are
declared.

  function keywords(c: psdl_component) return psdl_id_set;
  -- Returns an empty set if no keywords are specified.

  function requirements(c: psdl_component) return psdl_id_sequence;
  -- Returns an empty sequence if no requirements are specified.

  function informal_description(c: psdl_component) return text;
  -- Returns an empty string if no informal description is specified.

  function axioms(c: psdl_component) return text;
  -- Returns an empty string if no formal description is specified.

  ------------------------------------------------------------------
  --                      operations on psdl operators
--
  ------------------------------------------------------------------

  function inputs(o: operator) return type_declaration;
  -- Returns an empty type_declaration if no inputs are declared.

  function outputs(o: operator) return type_declaration;
  -- Returns an empty type_declaration if no outputs are declared.

  function states(o: operator) return type_declaration;
  -- Returns an empty type_declaration if no state variables are
declared.

  function initial_state(o: operator; v: variable) return expression;
  -- Raises initial_state_undefined if v is not initialized.

  function get_init_map(o: operator) return init_map;
  -- Returns an empty init_map if no initialization exists.

  function exceptions(o: operator) return psdl_id_set;
  -- Returns an empty set if no exceptions are declared.

  function specified_maximum_execution_time(o: operator) return
millisec;
  -- The maximum execution time specified in the specification of o.
  -- See also required_maximum_execution_time.
  -- Returns zero if no maximum execution time is declared.

  procedure add_input(stream: in psdl_id; t: in type_name; o: in out
operator);
  -- Adds a binding to the inputs map.
  -- Raises input_redeclared if stream is already in inputs(o).

  procedure add_output(stream: in psdl_id; t: in type_name; o: in out
operator);
  -- Adds a binding to the outputs map.
  -- Raises output_redeclared if stream is already in outputs(o).
```

```
  procedure add_state(stream: in psdl_id; t: in type_name; o: in out
operator);
  -- Adds a binding to the states map.
  -- Raises state_redeclared if stream is already in states(o).

  procedure add_initialization(stream: in psdl_id; e: in expression;
                               o: in out operator);
  -- Adds a binding to the init map.
  -- Raises initial_value_redeclared if stream is already bound in the
init map.

  procedure add_exception(e: psdl_id;
                          o: in out operator);
  -- Raises exception_redeclared if stream is already in exceptions(o).

  procedure set_specified_met(met: millisec; o: in out operator);
  -- Raises specified_met_redefined if specified_met is already defined.

  ------------------------------------------------------------------
  -- operations on all atomic psdl componets.                     --
  ------------------------------------------------------------------

  function ada_name(a: atomic_component) return ada_id;
     The name of the implementation module,
     usually the same as the name of the PSDL module (see function
na   .

  -- create an atomic operator
  function make_atomic_operator(psdl_name: psdl_id;
            ada_name: ada_id;
            gen_par: type_declaration := empty_type_declaration;
            keywords: psdl_id_set := empty;
            informal_description: text := empty;
            axioms: text := empty;
            input, output, state: type_declaration :=
empty_type_declaration;
            initialization_map: init_map := empty_init_map;
            exceptions: psdl_id_set := empty;
            specified_met: millisec := undefined_time;
            requirements: psdl_id_sequence := empty )
        return atomic_operator;

  -- create an atomic type
  function make_atomic_type(psdl_name: psdl_id;
            ada_name: ada_id;
            model: type_declaration;
            operations: operation_map;
            gen_par: type_declaration := empty_type_declaration;
            keywords: psdl_id_set := empty;
            informal_description, axioms: text := empty;
            requirements: psdl_id_sequence := empty )
        return atomic_type;

  ------------------------------------------------------------------
  -- operations on composite operators.
  ------------------------------------------------------------------

  function graph(co: composite_operator) return psdl_graph;

  function streams(co: composite_operator) return type_declaration;
  -- Returns an empty type_declaration if no local streams are declared.
```

```
  function timers(co: composite_operator) return psdl_id_set;
  -- Returns an empty set if no timers are declared.

  function get_trigger(component_op: op_id; co: composite_operator)
           return trigger;
  -- Returns the entire triggering condition for the specified component
operator.
  -- Derived from the control constraints.
  -- Raises no_trigger if component_op has no trigger.
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function get_trigger_type(component_op: op_id; co: composite_operator)
           return trigger_type;
  -- Returns the type of triggering condition for the specified
component operator.
  -- Derived from the control constraints, result is "none" if no
trigger.
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function execution_guard(component_op: op_id; co: composite_operator)
           return expression;
  -- Returns the if part of the triggering condition for the component
operator,
  -- "true" if no triggering condition is specified.
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function output_guard(component_op: op_id; output_stream: psdl_id;
                        co: composite_operator) return expression;
  -- Returns the if part of the output guard of the specified output
stream
  -- for the component operator,
  -- "true" if no output constraint with the stream is specified.
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function exception_trigger(component_op: op_id; exception_name:
psdl_id;
                             co: composite_operator) return expression;
  -- Returns the if part of the exception trigger for
  -- the component operator and exception name,
  -- "true" if there is an unconditional exception trigger
  -- in the control contraints, "false" if no exception
  -- trigger is specified for component_op in the control constraints.
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function timer_operations(component_op: op_id; co: composite_operator)
           return timer_op_set;
  -- Returns the timer_op part of the control constraint for the
  -- component operator, "none" if no timer operation is specified.
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function timer_op_guard(component_op: op_id; timer_id: psdl_id;
                          op_id: timer_op_id;
                          co: composite_operator) return expression;
  -- Returns the the timer operation guard of the specified timer and
```

```
   -- timer operation for the component operator,
   -- "true" if no timer operation guard is specified.
   -- Raises null_component if co is null.
   -- Raises not_an_operator if co is a psdl type.
   -- Raises not_a_composite_component if co is atomic.
   -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

   function period(component_op: op_id; co: composite_operator)
            return millisec;
   -- Returns the period part of the control constraint for the
   -- component operator, zero if no period is specified.
   -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

   function finish_within(component_op: op_id; co: composite_operator)
            return millisec;
   -- Returns the finish_within part of the control constraint for the
   -- component operator, zero if no finish_within is specified.
   -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

   function minimum_calling_period(component_op: op_id; co:
composite_operator)
            return millisec;
   -- Returns the minimum calling period part of the control constraint
for the
   -- component operator, zero if no minimum calling period is specified.
   -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

   function maximum_response_time(component_op: op_id; co:
composite_operator)
            return millisec;
   -- Returns the maximum_response_time part of the control constraint
for the
   -- component operator, zero if no maximum_response_time is specified.
   -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

   function required_maximum_execution_time
            (component_op: op_id; co: composite_operator) return
millisec;
   -- Returns the maximum execution time part of the control constraint
for the
   -- component operator, zero if no maximum execution time is specified
   -- in the graph. This includes time used by the implementations
   -- of the control constraints and stream operations, and should be
   -- greater than or equal to the specified_maximum_execution time for
   -- the component operator if it is defined(greater than zero).
   -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

   function latency(producer_op, consumer_op: op_id; stream_name:
psdl_id;
                    co: composite_operator) return millisec;
   -- Returns the timing label on the edge from the producer operator
   -- to the consumer operator in the graph, zero if none.
   -- Represents the maximum data transmission delay allowed for
   -- the data stream, for modeling network delay in distributed systems.
```

```
  -- Raises not_a_subcomponent if component_op is not a vertex in
graph(co).

  function implementation_description(co: composite_operator)
     return text;

  -- Create a composite operator.
  function make_composite_operator(name: psdl_id;
            gen_par: type_declaration := empty_type_declaration;
            keywords: psdl_id_set := empty;
            informal_description: text := empty;
            axioms: text := empty;
            input, output, state: type_declaration :=
empty_type_declaration;
            initialization_map: init_map := empty_init_map;
            exceptions: psdl_id_set := empty;
            specified_met: millisec := undefined_time;
            graph: psdl_graph := empty_psdl_graph;
            streams: type_declaration := empty_type_declaration;
            timers: psdl_id_set := empty;
            trigger: trigger_map := empty_trigger_map;
            exec_guard: exec_guard_map := empty_exec_guard_map;
            out_guard: out_guard_map := empty_out_guard_map;
            excep_trigger: excep_trigger_map := empty_excep_trigger_map;
            timer_op: timer_op_map := empty_timer_op_map;
            per, fw, mcp, mrt: timing_map := empty_timing_map;
            impl_desc: text:= empty;
            requirements: psdl_id_sequence := empty)
         return composite_operator;

  procedure add_vertex(opname: in op_id; co: in out composite_operator;
                       met: in millisec := undefined_time);

  procedure add_edge(x, y: in op_id; stream: in psdl_id;
                     co: in out composite_operator;
                     latency: in millisec := undefined_time);

  procedure add_stream(s: in psdl_id; t: in type_name;
                       co: in out composite_operator);

  procedure remove_stream(s: in psdl_id; co: in out composite_operator);

  procedure add_timer(t: in psdl_id; co: in out composite_operator);

  procedure set_graph(g: in psdl_graph; co: in out composite_operator);

  procedure set_trigger_type(o: in op_id; t: in trigger_type;
                             co: in out composite_operator);

  procedure set_trigger(o: in op_id; t: in trigger;
                        co: in out composite_operator);

  procedure set_execution_guard(o: in op_id; e: in expression;
                                co: in out composite_operator);

  procedure set_output_guard(o: in op_id; stream: in psdl_id;
                             e: in expression; co: in out
composite_operator);

  procedure set_exception_trigger(o: in op_id; excep: in psdl_id;
                                  e: in expression;
```

```
                                        co: in out composite_operator);

  procedure add_timer_op(o: op_id; timer_id: in psdl_id; top: in
timer_op_id;
                         e: in expression; co: in out
composite_operator);

  procedure set_period(o: in op_id; p: in millisec;
                       co: in out composite_operator);
  -- Raises period_redefined if the period is already defined.


  procedure set_finish_within(o: in op_id; fw: in millisec;
                              co: in out composite_operator);
  -- Raises finish_within_redefined if the finish_within is already
defined.

  procedure set_minimum_calling_period(o: in op_id; mcp: in millisec;
                                       co: in out composite_operator);
  -- Raises minimum_calling_period_redefined if the
  -- minimum_calling_period is already defined.

  procedure set_maximum_response_time(o: in op_id; mrt: in millisec;
                                      co: in out composite_operator);
  -- Raises maximum_response_time_redefined if the
  -- maximum_response_time is already defined.

  ------------------------------------------------------------------------
-
  -- operations on all psdl types.
  ------------------------------------------------------------------------
-

  function model(t: data_type) return type_declaration;
  -- Returns the conceptual representation declared in the specification
part,
  -- empty if not specified.

  function operations(t: data_type) return operation_map;
  -- Returns an environment binding operation names to operation
definitions,
  -- an empty map if the type does not define any operations.

  ------------------------------------------------------------------------
-
  -- operations on composite psdl data types.
  ------------------------------------------------------------------------
-

  function data_structure(t: composite_type) return type_name;
  -- Returns the data structure declared in the psdl implementation
part.
  -- Raises no_data_structure if the type is implemented in Ada.

  -- Create a composite type.
  function make_composite_type(name: psdl_id;
           model: type_declaration;
           data_structure: type_name;
           operations: operation_map;
           gen_par: type_declaration := empty_type_declaration;
           keywords: psdl_id_set := empty;
```

```
            informal_description, axioms: text := empty;
            requirements: psdl_id_sequence := empty )
    return composite_type;

  -- System functions

  procedure recycle(c: in out psdl_component);
    -- Sets c to the null_component and recycles its storage.

private
  type psdl_component_record(category: component_type;
                             granularity: implementation_type) is
    record
      name: psdl_id;
      gen_par: type_declaration;
      keyw: psdl_id_set;
      inf_desc, ax: text;
      reqm: psdl_id_sequence;
      case category is
        when psdl_operator =>
          input, output, state: type_declaration;
          init: init_map;
          excep: psdl_id_set;
          smet: millisec;
          case granularity is
            when atomic => o_ada_name: ada_id;
            when composite =>
              g: psdl_graph;
              str: type_declaration;
              tim: psdl_id_set;
              trig: trigger_map;
              eg: exec_guard_map;
              og: out_guard_map;
              et: excep_trigger_map;
              tim_op: timer_op_map;
              per, fw, mcp, mrt: timing_map;
              impl_desc: text;  -- Description in the implementation
part.
          end case;
        when psdl_type =>
          mdl: type_declaration;
          ops: operation_map;
          case granularity is
            when atomic => t_ada_name: ada_id;
            when composite => data_str: type_name;
          end case;
      end case;
  end record;
  -- Invariant: a psdl component contains the only references
  -- to all of its subcomponents of type set{t} or map{t},
  -- for all types t.
  -- The assign operation should be used on those components
  -- whenever these components are passed in or out by the operations
  -- becuase it makes copies, thus preserving the invariant.
  -- This makes it safe to recycle a psdl component
  -- when the last reference is gone,
  -- because the subcomponents cannot be shared with anything else.
end psdl_component_pkg;
```

```
package body psdl_component_pkg is  use operation_map_pkg ;
  function eq ( x  ,
y :  psdl_component   ) return boolean is
begin   return ( x . name . s  = y . name . s   )  ;
   end eq;
  function empty_operation_map  return operation_map is  m  :
operation_map  ;

begin   create ( null_component    ,
m   );
   return m  ;
   end empty_operation_map;
  function component_category ( c  :  psdl_component   ) return
component_type is
begin
if c  = null_component
then   raise undefined_component;
    end if;
   return c . category  ;
   end component_category;
  function component_granularity ( c  :  psdl_component   ) return
implementation_type is
begin
if c  = null_component
then   raise undefined_component;
    end if;
   return c . granularity  ;
   end component_granularity;
  function name ( c  :  psdl_component   ) return psdl_id is
begin
if c  = null_component
then   raise undefined_component;
    end if;
   return c . name  ;
   end name;
  function generic_parameters ( c  :  psdl_component   ) return
type_declaration is  result  : type_declaration  ;

begin
if c  = null_component
then   raise undefined_component;
    end if;
   assign ( result    ,
c . gen_par   );
   return result  ;
   end generic_parameters;
  function keywords ( c  :  psdl_component   ) return psdl_id_set is
result  : psdl_id_set  ;

begin
if c  = null_component
then   raise undefined_component;
    end if;
   assign ( result    ,
c . keyw   );
   return result  ;
   end keywords;

  function requirements( c  :  psdl_component   ) return
psdl_id_sequence is  result  : psdl_id_sequence  ;
```

```
begin
if c = null_component
then raise undefined_component;
    end if;
   assign ( result , c . reqm );
   return result;
   end requirements;
  function informal_description ( c  :  psdl_component   ) return text
is
begin
if c  = null_component
then   raise undefined_component;
    end if;
   return c . inf_desc  ;
   end informal_description;
  function axioms ( c  :  psdl_component   ) return text is
begin
if c  = null_component
then   raise undefined_component;
    end if;
   return c . ax  ;
   end axioms;
  function inputs ( o  :  operator   ) return type_declaration is
result  : type_declaration  ;

begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

else   assign ( result    ,
o . input   );
   return result  ;
  end if;
   end inputs;
  function outputs ( o  :  operator   ) return type_declaration is
result  : type_declaration  ;

begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

else   assign ( result    ,
o . output   );
   return result  ;
  end if;
   end outputs;
  function states ( o  :  operator   ) return type_declaration is
result  : type_declaration  ;

begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

else   assign ( result    ,
o . state   );
```

```
    return result  ;
   end if;
    end states;
   function initial_state ( o  :  operator  ;
 v  :  variable  ) return expression is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif not member ( v    ,
o . init   )  then   raise initial_state_undefined;

else   return fetch ( o . init    ,
v   )  ;
  end if;
   end initial_state;
  function get_init_map ( o  :  operator   ) return init_map is  result
: init_map  ;

begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

else   assign ( result    ,
o . init   );
   return result  ;
  end if;
   end get_init_map;
  function exceptions ( o  :  operator   ) return psdl_id_set is  result
: psdl_id_set  ;

begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

else   assign ( result    ,
o . excep   );
   return result  ;
  end if;
   end exceptions;
  function specified_maximum_execution_time ( o  :  operator   ) return
millisec is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

else   return o . smet  ;
  end if;
   end specified_maximum_execution_time;
  procedure add_input ( stream  : in psdl_id  ;
 t  : in type_name ;
 o  : in out operator  ) is
begin
```

```
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif member ( stream    ,
o . input   )   then   raise input_redeclared;

else   bind ( stream    ,
t   ,
o . input   );
  end if;
   end add_input;
  procedure add_output ( stream  : in psdl_id  ;
 t  : in type_name ;
 o  : in out operator  ) is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif member ( stream    ,
o . output   )   then   raise output_redeclared;

else   bind ( stream    ,
t   ,
o . output   );
  end if;
   end add_output;
  procedure add_state ( stream  : in psdl_id  ;
 t  : in type_name ;
 o  : in out operator  ) is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif member ( stream    ,
o . state   )   then   raise state_redeclared;

else   bind ( stream    ,
t   ,
o . state   );
  end if;
   end add_state;
  procedure add_initialization ( stream  : in psdl_id  ;
 e  : in expression ;
 o  : in out operator  ) is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif member ( stream    ,
o . init   )   then   raise initial_value_redeclared;

else   bind ( stream    ,
e   ,
```

```
o . init   );
  end if;
   end add_initialization;
  procedure add_exception ( e  :  psdl_id  ;
 o  : in out operator  ) is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif member ( e    ,
o . excep   )   then   raise exception_redeclared;

else   add ( e    ,
o . excep   );
  end if;
   end add_exception;
  procedure set_specified_met ( met  :  millisec  ;
 o  : in out operator  ) is
begin
if o  = null_component
then   raise undefined_component;

elsif o . category  /= psdl_operator  then   raise not_an_operator;

elsif o . smet  /= undefined_time  then   raise input_redeclared;

else   o . smet := met  ;
  end if;
   end set_specified_met;
  function ada_name ( a  :  atomic_component   ) return ada_id is
begin
if a  = null_component
then   raise undefined_component;
    end if;
   case a . granularity   is  when atomic   =>   case a . category   is
when psdl_operator   =>   return a . o_ada_name  ;
   when psdl_type   =>   return a . t_ada_name  ;
  end case;
   when composite   =>   raise not_an_atomic_component;
  end case;
   end ada_name;
  function make_atomic_operator ( psdl_name  :  psdl_id  ;
 ada_name  :  ada_id ;
 gen_par  :  type_declaration := empty_type_declaration  ;
 keywords  :  psdl_id_set := empty  ;
 informal_description  :  text := empty  ;
 axioms  :  text := empty  ;
 input  ,
output ,
state :  type_declaration := empty_type_declaration  ;
 initialization_map  :  init_map := empty_init_map  ;
 exceptions  :  psdl_id_set := empty  ;
 specified_met  :  millisec := undefined_time ;
 requirements : psdl_id_sequence := empty  ) return atomic_operator is
x  : atomic_operator  ;

begin   x := new psdl_component_record ( category   => psdl_operator
,
granularity   => atomic   )  ;
```

```
   x . name := psdl_name  ;
   x . o_ada_name := ada_name  ;
   x . gen_par := gen_par  ;
   x . keyw := keywords  ;
   x . inf_desc := informal_description  ;
   x . reqm := requirements ;
   x . ax := axioms  ;
   x . input := input  ;
   x . output := output  ;
   x . state := state  ;
   x . init := initialization_map  ;
   x . excep := exceptions  ;
   x . smet := specified_met  ;
   return x  ;
   end make_atomic_operator;
  function make_atomic_type ( psdl_name  :  psdl_id  ;
 ada_name  :  ada_id ;
 model  :  type_declaration ;
 operations  :  operation_map ;
 gen_par  :  type_declaration := empty_type_declaration  ;
 keywords  :  psdl_id_set := empty  ;
 informal_description  ,
axioms :  text := empty;
 requirements : psdl_id_sequence := empty ) return atomic_type is  x  :
atomic_type  ;

begin   x := new psdl_component_record ( category   => psdl_type    ,
granularity   => atomic   )  ;
   x . name := psdl_name  ;
   x . t_ada_name := ada_name  ;
   x . mdl := model  ;
   x . ops := operations  ;
   x . gen_par := gen_par  ;
   x . keyw := keywords  ;
   x . inf_desc := informal_description  ;
   x . reqm := requirements ;
   x . ax := axioms  ;
   return x  ;
   end make_atomic_type;
  function graph ( co  :  composite_operator   ) return psdl_graph is
result  : psdl_graph  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   assign ( result    ,
co . g   );
   return result  ;
   end graph;
  function streams ( co  :  composite_operator   ) return
type_declaration is  result  : type_declaration  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
```

```
   assign ( result     ,
co . str    );
   return result  ;
   end streams;
  function timers ( co   :  composite_operator    ) return psdl_id_set is
result  : psdl_id_set  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   assign ( result     ,
co . tim    );
   return result  ;
   end timers;
  function get_trigger ( component_op   :  op_id  ;
 co  :  composite_operator  ) return trigger is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif not has_vertex ( component_op     ,
co . g    )  then   raise not_a_subcomponent;

else   return fetch ( co . trig     ,
component_op    )  ;
  end if;
   end get_trigger;
  function get_trigger_type ( component_op   :  op_id  ;
 co  :  composite_operator  ) return trigger_type is  t_record  :
trigger  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif not has_vertex ( component_op     ,
co . g    )  then   raise not_a_subcomponent;

elsif ( not member ( component_op     ,
co . trig    )   )   then   return by_none  ;

else   t_record := fetch ( co . trig     ,
component_op    )  ;
   return t_record . tt  ;
  end if;
   end get_trigger_type;
  function execution_guard ( component_op   :  op_id  ;
 co  :  composite_operator  ) return expression is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
```

```
elsif not has_vertex ( component_op     ,
co . g    )  then    raise not_a_subcomponent;

elsif ( not member ( component_op     ,
co . eg    )    )    then    return true_expression  ;

else    return fetch ( co . eg     ,
component_op    )  ;
  end if;
   end execution_guard;
  function output_guard ( component_op   :  op_id  ;
 output_stream   :  psdl_id ;
 co   :  composite_operator  ) return expression is  temp_id  : output_id
;

begin
if co  = null_component
then    raise undefined_component;

elsif co . category  /= psdl_operator  then    raise not_an_operator;
   end if;

if not has_vertex ( component_op     ,
co . g    )
then    raise not_a_subcomponent;
    end if;
   temp_id . op := component_op  ;
   temp_id . stream := output_stream  ;

if ( not member ( temp_id     ,
co . og    )    )
then    return true_expression  ;

else    return fetch ( co . og     ,
temp_id    )  ;
  end if;
   end output_guard;
  function exception_trigger ( component_op   :  op_id  ;
 exception_name   :  psdl_id ;
 co   :  composite_operator  ) return expression is  temp_id  : excep_id
;

begin
if co  = null_component
then    raise undefined_component;

elsif co . category  /= psdl_operator  then    raise not_an_operator;

elsif not has_vertex ( component_op     ,
co . g    )  then    raise not_a_subcomponent;
   end if;
   temp_id . op := component_op  ;
   temp_id . excep := exception_name  ;

if ( not member ( temp_id     ,
co . et    )    )
then    return false_expression  ;

else    return fetch ( co . et     ,
temp_id    )  ;
  end if;
```

```
   end exception_trigger;
  function timer_operations ( component_op  :  op_id  ;
 co  :  composite_operator  ) return timer_op_set is  result  :
timer_op_set  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;

else   assign ( result    ,
fetch ( co . tim_op    ,
component_op   )   );
   return result  ;
  end if;
   end timer_operations;
  function timer_op_guard ( component_op  :  op_id  ;
 timer_id  :  psdl_id ;
 op_id  :  timer_op_id ;
 co  :  composite_operator  ) return expression is  top_set  :
timer_op_set  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;

else   top_set := fetch ( co . tim_op    ,
component_op   )  ;
  end if;
   declare -- begin generator loop
generator_loop_return_value: expression;
return_from_generator_loop: exception;
exit_from_generator_loop: exception;
procedure generator_loop_body(top: timer_op) is
begin

if eq ( top . timer_id    ,
timer_id   )   and then top . op_id  = op_id
then  generator_loop_return_value := top . guard  ;
raise return_from_generator_loop;
   end if;
end generator_loop_body;
procedure execute_generator_loop is new timer_op_set_pkg .
scan(generator_loop_body);
begin
execute_generator_loop(top_set  );
exception
when exit_from_generator_loop => null;
when return_from_generator_loop => return generator_loop_return_value;
end; -- of generator loop
   return true_expression  ;
   end timer_op_guard;
```

```
  function period ( component_op  :  op_id  ;
 co  :  composite_operator  ) return millisec is
begin
if co  = null_component
then   raise undefined_component;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;

else   return fetch ( co . per    ,
component_op   )  ;
  end if;
   end period;
  function finish_within ( component_op  :  op_id  ;
 co  :  composite_operator  ) return millisec is
begin
if co  = null_component
then   raise undefined_component;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;

else   return fetch ( co . fw    ,
component_op   )  ;
  end if;
   end finish_within;
  function minimum_calling_period ( component_op  :  op_id  ;
 co  :  composite_operator  ) return millisec is
begin
if co  = null_component
then   raise undefined_component;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;

else   return fetch ( co . mcp    ,
component_op   )  ;
  end if;
   end minimum_calling_period;
  function maximum_response_time ( component_op  :  op_id  ;
 co  :  composite_operator  ) return millisec is
begin
if co  = null_component
then   raise undefined_component;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;

else   return fetch ( co . mrt    ,
component_op   )  ;
  end if;
   end maximum_response_time;
  function required_maximum_execution_time ( component_op  :  op_id  ;
 co  :  composite_operator  ) return millisec is
begin
if co  = null_component
then   raise undefined_component;

elsif not has_vertex ( component_op    ,
co . g   )  then   raise not_a_subcomponent;
```

```
else   return maximum_execution_time ( component_op     ,
co . g    )  ;
  end if;
   end required_maximum_execution_time;
  function latency ( producer_op  ,
consumer_op :  op_id  ;
 stream_name  :  psdl_id ;
 co  :  composite_operator  ) return millisec is
begin
if co  = null_component
then   raise undefined_component;

elsif not has_vertex ( producer_op     ,
co . g    )  or not has_vertex ( consumer_op     ,
co . g    )  then   raise not_a_subcomponent;

else   return latency ( producer_op     ,
consumer_op    ,
stream_name    ,
co . g    )  ;
  end if;
   end latency;
  function implementation_description ( co  :  composite_operator    )
return text is
begin
if co  = null_component
then   raise undefined_component;
    end if;
   return co . impl_desc  ;
   end implementation_description;
  function make_composite_operator ( name  :  psdl_id  ;
 gen_par  :  type_declaration := empty_type_declaration  ;
 keywords  :  psdl_id_set := empty  ;
 informal_description  :  text := empty  ;
 axioms  :  text := empty  ;
 input  ,
output ,
state :  type_declaration := empty_type_declaration  ;
 initialization_map  :  init_map := empty_init_map  ;
 exceptions  :  psdl_id_set := empty  ;
 specified_met  :  millisec := undefined_time  ;
 graph  :  psdl_graph := empty_psdl_graph  ;
 streams  :  type_declaration := empty_type_declaration  ;
 timers  :  psdl_id_set := empty  ;
 trigger  :  trigger_map := empty_trigger_map  ;
 exec_guard  :  exec_guard_map := empty_exec_guard_map  ;
 out_guard  :  out_guard_map := empty_out_guard_map  ;
 excep_trigger  :  excep_trigger_map := empty_excep_trigger_map  ;
 timer_op  :  timer_op_map := empty_timer_op_map  ;
 per  ,
fw ,
mcp ,
mrt :  timing_map := empty_timing_map  ;
 impl_desc  :  text := empty ;
requirements : psdl_id_sequence := empty    ) return composite_operator
is  x  : composite_operator  ;

begin   x := new psdl_component_record ( category   => psdl_operator
,
granularity   => composite    )  ;
   x . name := name  ;
```

```
    x . gen_par := gen_par  ;
    x . keyw := keywords  ;
    x . inf_desc := informal_description  ;
    x . reqm := requirements ;
    x . ax := axioms  ;
    x . input := input  ;
    x . output := output  ;
    x . state := state  ;
    x . init := initialization_map  ;
    x . excep := exceptions  ;
    x . smet := specified_met  ;
    x . g := graph  ;
    x . str := streams  ;
    x . tim := timers  ;
    x . trig := trigger  ;
    x . eg := exec_guard  ;
    x . og := out_guard  ;
    x . et := excep_trigger  ;
    x . tim_op := timer_op  ;
    x . per := per  ;
    x . fw := fw  ;
    x . mcp := mcp  ;
    x . mrt := mrt  ;
    x . impl_desc := impl_desc  ;
    return x  ;
    end make_composite_operator;
  procedure add_vertex ( opname  : in op_id  ;
 co  : in out composite_operator ;
 met  : in millisec := undefined_time   ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
    end if;
    co . g := add_vertex ( opname    ,
co . g   ,
met   )  ;
    end add_vertex;
  procedure add_edge ( x  ,
y : in op_id  ;
 stream  : in psdl_id ;
 co  : in out composite_operator ;
 latency  : in millisec := undefined_time   ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
    end if;
    co . g := add_edge ( x    ,
y   ,
stream   ,
co . g   ,
latency   )  ;
    end add_edge;
  procedure add_stream ( s  : in psdl_id  ;
 t  : in type_name ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
```

```
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   bind ( s    ,
t   ,
co . str   );
   end add_stream;
  procedure remove_stream ( s   : in psdl_id  ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   remove ( s    ,
co . str   );
   end remove_stream;
  procedure add_timer ( t   : in psdl_id  ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   add ( t    ,
co . tim   );
   end add_timer;
  procedure set_graph ( g   : in psdl_graph  ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;
    end if;
   co . g := g  ;
   end set_graph;
  procedure set_trigger_type ( o   : in op_id  ;
 t  : in trigger_type ;
 co  : in out composite_operator  ) is  t_record  : trigger  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   t_record . tt := t  ;
   t_record . streams := empty  ;
   bind ( o    ,
t_record   ,
co . trig   );
   end set_trigger_type;
  procedure set_trigger ( o   : in op_id  ;
 t  : in trigger ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;
```

```
elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   bind ( o    ,
t   ,
co . trig   );
   end set_trigger;
  procedure set_execution_guard ( o  : in op_id  ;
 e  : in expression ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   bind ( o    ,
e   ,
co . eg   );
   end set_execution_guard;
  procedure set_output_guard ( o  : in op_id  ;
 stream  : in psdl_id ;
 e  : in expression ;
 co  : in out composite_operator  ) is  temp_id  : output_id  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   temp_id . op := o  ;
   temp_id . stream := stream  ;
   bind ( temp_id    ,
e   ,
co . og   );
   end set_output_guard;
  procedure set_exception_trigger ( o  : in op_id  ;
 excep  : in psdl_id ;
 e  : in expression ;
 co  : in out composite_operator  ) is  temp_id  : excep_id  ;

begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   temp_id . op := o  ;
   temp_id . excep := excep  ;
   bind ( temp_id    ,
e   ,
co . et   );
   end set_exception_trigger;
  procedure add_timer_op ( o  :  op_id  ;
 timer_id  : in psdl_id ;
 top  : in timer_op_id ;
 e  : in expression ;
 co  : in out composite_operator  ) is  temp_id  : timer_op  ;
  temp_set  : timer_op_set  ;

begin
```

```
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;
   end if;
   temp_id . op_id := top  ;
   temp_id . timer_id := timer_id  ;
   temp_id . guard := e  ;
   temp_set := fetch ( co . tim_op    ,
o   )  ;
   add ( temp_id    ,
temp_set   );
   bind ( o    ,
temp_set   ,
co . tim_op   );
   end add_timer_op;
  procedure set_period ( o  : in op_id  ;
 p  : in millisec ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif ( fetch ( co . per    ,
o   )    )  /= undefined_time  then   raise period_redefined;
   end if;
   bind ( o    ,
p   ,
co . per   );
   end set_period;
  procedure set_finish_within ( o  : in op_id  ;
 fw  : in millisec ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif ( fetch ( co . fw    ,
o   )    )  /= undefined_time  then   raise finish_within_redefined;
   end if;
   bind ( o    ,
fw   ,
co . fw   );
   end set_finish_within;
  procedure set_minimum_calling_period ( o  : in op_id  ;
 mcp  : in millisec ;
 co  : in out composite_operator  ) is
begin
if co  = null_component
then   raise undefined_component;

elsif co . category  /= psdl_operator  then   raise not_an_operator;

elsif ( fetch ( co . mcp    ,
o   )    )  /= undefined_time  then   raise
minimum_calling_period_redefined;
   end if;
```

```
   bind ( o     ,
mcp    ,
co . mcp    );
   end set_minimum_calling_period;
  procedure set_maximum_response_time ( o   : in op_id   ;
 mrt   : in millisec ;
 co   : in out composite_operator   ) is
begin
if co   = null_component
then    raise undefined_component;

elsif co . category   /= psdl_operator   then    raise not_an_operator;

elsif ( fetch ( co . mrt     ,
o    )     )   /= undefined_time   then    raise
maximum_response_time_redefined;
   end if;
   bind ( o     ,
mrt    ,
co . mrt    );
   end set_maximum_response_time;
  function model ( t   :   data_type    ) return type_declaration is
result   : type_declaration   ;

begin
if t   = null_component
then    raise undefined_component;
    end if;
   case t . category    is   when psdl_operator    =>    raise not_a_type;
   when psdl_type    =>    assign ( result     ,
t . mdl    );
   return result   ;
  end case;
   end model;
  function operations ( t   :   data_type    ) return operation_map is
result   : operation_map   ;

begin
if t   = null_component
then    raise undefined_component;
    end if;
   case t . category    is   when psdl_operator    =>    raise not_a_type;
   when psdl_type    =>    assign ( result     ,
t . ops    );
   return result   ;
  end case;
   end operations;
  function data_structure ( t   :   composite_type    ) return type_name is
result   : type_name   ;

begin
if t   = null_component
then    raise undefined_component;
    end if;
   case t . category    is   when psdl_operator    =>    raise not_a_type;
   when psdl_type    =>    case t . granularity    is   when atomic    =>
raise no_data_structure;
   when composite    =>    result := new type_name_record   ;
   result . name := t . data_str . name   ;
   assign ( result . formals     ,
t . data_str . formals    );
```

```
   assign ( result . gen_par     ,
t . data_str . gen_par   );
   return result  ;
  end case;
  end case;
   end data_structure;
  function make_composite_type ( name   :  psdl_id  ;
 model   :  type_declaration ;
 data_structure   :  type_name ;
 operations   :  operation_map ;
 gen_par   :  type_declaration := empty_type_declaration  ;
 keywords   :  psdl_id_set := empty  ;
 informal_description  ,
axioms :  text := empty ;
requirements : psdl_id_sequence := empty ) return composite_type is  x
: composite_type  ;

begin   x := new psdl_component_record ( category   => psdl_type     ,
granularity   => composite   )  ;
   x . name := name  ;
   x . gen_par := gen_par  ;
   x . keyw := keywords  ;
   x . inf_desc := informal_description  ;
   x . reqm := requirements ;
   x . ax := axioms  ;
   x . mdl := model  ;
   x . ops := operations  ;
   x . data_str := data_structure  ;
   return x  ;
   end make_composite_type;
  procedure recycle ( c  : in out psdl_component   ) is
begin
if c  /= null_component
then   recycle ( c . gen_par    );
   recycle ( c . keyw    );
   case c . category   is  when psdl_operator   =>   recycle ( c . input
);
   recycle ( c . output    );
   recycle ( c . state    );
   recycle ( c . init    );
   recycle ( c . excep    );
   case c . granularity   is  when atomic   =>   null;
   when composite   =>   recycle ( c . g    );
   recycle ( c . str    );
   recycle ( c . tim    );
   recycle ( c . trig    );
   recycle ( c . eg    );
   recycle ( c . og    );
   recycle ( c . et    );
   recycle ( c . tim_op    );
   recycle ( c . per    );
   recycle ( c . fw    );
   recycle ( c . mcp    );
   recycle ( c . mrt    );
  end case;
   when psdl_type   =>   recycle ( c . mdl    );
   recycle ( c . ops    );
   case c . granularity   is  when atomic   =>   null;
   when composite   =>   recycle ( c . data_str . gen_par    );
  end case;
  end case;
```

```
c := null_component   ;
 end if;
end recycle;
end psdl_component_pkg;
```

APPENDIX H. AYACC SOURCE CODE LISTING

```
-- $Header:
/export/home/berzins/CAPS/build/R2.1/PSDL_TYPE/RCS/parser.y,v 1.7
1996/08/29 23:49:02 berzins Exp berzins $
------------------------------------------------------------------------
-
-- This file is the ayacc source file for PSDL parser.
------------------------------------------------------------------------
-

-- token declarations section

%token  '(' ')' ','   '[' ']' ':'  '.' '|'
%token ARROW
%token TRUE FALSE
%token ADA_TOKEN AXIOMS_TOKEN
%token BY_ALL_TOKEN REQ_BY_TOKEN BY_SOME_TOKEN
%token CALL_PERIOD_TOKEN CONTROL_TOKEN
%token CONSTRAINTS_TOKEN
%token DATA_TOKEN DESCRIPTION_TOKEN
%token EDGE_TOKEN END_TOKEN EXCEPTIONS_TOKEN
%token EXCEPTION_TOKEN EXECUTION_TOKEN
%token FINISH_TOKEN
%token GENERIC_TOKEN GRAPH_TOKEN
%token HOURS_TOKEN
%token IF_TOKEN IMPLEMENTATION_TOKEN
%token INITIALLY_TOKEN INPUT_TOKEN
%token KEYWORDS_TOKEN
%token MAXIMUM_TOKEN MINIMUM_TOKEN
%token MICROSEC_TOKEN
%token MIN_TOKEN MS_TOKEN  MOD_TOKEN
%token NOT_TOKEN
%token OPERATOR_TOKEN OR_TOKEN OUTPUT_TOKEN
%token PERIOD_TOKEN PROPERTY_TOKEN
%token RESET_TOKEN RESPONSE_TOKEN
%token SEC_TOKEN SPECIFICATION_TOKEN
%token START_TOKEN STATES_TOKEN STOP_TOKEN
%token STREAM_TOKEN
%token TIME_TOKEN
%token TIMER_TOKEN TRIGGERED_TOKEN TYPE_TOKEN
%token VERTEX_TOKEN
%token WITHIN_TOKEN
%token IDENTIFIER
%token INTEGER_LITERAL REAL_LITERAL
%token STRING_LITERAL
%token TEXT_TOKEN

-- operator precedences
-- left means group and evaluate from the left
%left AND_TOKEN OR_TOKEN XOR_TOKEN LOGICAL_OPERATOR
%left '<' '>' '=' GREATER_THAN_OR_EQUAL LESS_THAN_OR_EQUAL INEQUALITY
RELATIONAL_OPERATOR
%left '+' '-' '&' BINARY_ADDING_OPERATOR
%left UNARY_ADDING_OPERATOR
%left '*'  '/'  MOD_TOKEN REM_TOKEN MULTIPLYING_OPERATOR
%left EXP_TOKEN ABS_TOKEN NOT_TOKEN HIGHEST_PRECEDENCE_OPERATOR
```

```
%start start_symbol  -- this is an artificial start symbol, for
initialization


%with psdl_concrete_type_pkg, expression_pkg;
%use psdl_concrete_type_pkg, expression_pkg;

-- declaration of the value type for the parser stack.
{
    type token_category_type is (integer_cat,
                                 text_cat,
                                 psdl_id_cat,
                                 psdl_id_sequence_cat,
                                 op_id_cat,
                                 operator_name_cat,
                                 opt_arg_cat,
                                 type_name_cat,
                          type_decl_cat,
                                 timer_op_id_cat,
                                 expression_cat,
                                 expression_seq_cat,
                                 property_map_cat,
                                 no_value_cat);

    type yystype (token_category: token_category_type := no_value_cat)
is
      record
         case token_category is
           -- lexical token attributes:
           when integer_cat =>
             integer_value: integer;
           when text_cat =>
             text_value: text;
           -- grammar psdl_id attributes:
           when psdl_id_cat =>
             psdl_id_value: psdl_id;
           when psdl_id_sequence_cat =>
             psdl_id_sequence_value: psdl_id_sequence;
           when op_id_cat =>
             op_id_value: op_id;
           when operator_name_cat =>
             type_name_part, op_name_part: psdl_id;
           when opt_arg_cat =>
             input_value, output_value: psdl_id_sequence;
           when type_name_cat =>
             type_name_value: type_name;
           when type_decl_cat =>
             type_decl_value: type_declaration;
          when timer_op_id_cat =>
             timer_op_id_value: timer_op_id;
           when expression_cat =>
             expression_value: expression;
           when expression_seq_cat =>
             expression_seq_value: expression_sequence;
           when property_map_cat =>
             property_map_value: init_map;
           when no_value_cat => null;
         end case;
       end record;
}
```

```
%%

start_symbol
        :   { the_program := empty_psdl_program; }
          psdl
        ;

psdl    : psdl component
            { if member(name(the_component), the_program)
              then yyerror("Component redefined: " &
convert(name(the_component)));
              else bind(name(the_component), the_component,
the_program);
              end if; }
        |
        ;

component
        : data_type
        | operator
        ;

data_type
        : TYPE_TOKEN IDENTIFIER
            { the_operation_map := empty_operation_map;
              is_specification := true;  }
          type_spec
            { is_specification := false; }
          type_impl
            { -- Construct the psdl type using global variables.
              build_psdl_type($2.psdl_id_value,
                              the_ada_name,
                              the_model,
                              the_data_structure,
                              the_operation_map,
                              the_type_gen_par,
                              the_keywords,
                              the_description,
                              the_axioms,
                              is_atomic_type,
                              the_component,
                              the_reqmts_trace);
                  output_trace(the_component);
           }
        ;

type_spec
        :
          SPECIFICATION_TOKEN optional_generic_param optional_type_decl
          op_spec_list functionality END_TOKEN
        ;

optional_generic_param
        : GENERIC_TOKEN
            { the_type_decl := empty_type_declaration; }
          list_of_type_decl
            { the_type_gen_par := the_type_decl; }
        |   { the_type_gen_par := empty_type_declaration; }
        ;

optional_type_decl
```

```
        :   { the_type_decl := empty_type_declaration; }
          list_of_type_decl
            { the_model := the_type_decl; }
        |   { the_model := empty_type_declaration; }
        ;

op_spec_list
        : op_spec_list OPERATOR_TOKEN IDENTIFIER operator_spec
            { build_psdl_operator($3.psdl_id_value,
                                  ada_id($3.psdl_id_value),
                                  the_gen_par,
                                  the_keywords,
                                  the_description,
                                  the_axioms,
                                  the_input,
                                  the_output,
                                  the_state,
                                  the_initial_expression_map,
                                  the_exceptions,
                                  the_specified_met,
                                  empty_psdl_graph,
                                  empty_type_declaration,
                                  empty,
                                  empty_trigger_map,
                                  empty_exec_guard_map,
                                  empty_out_guard_map,
                                  empty_excep_trigger_map,
                                  empty_timer_op_map,
                                  empty_timing_map,
                                  empty_timing_map,
                                  empty_timing_map,
                                  empty_timing_map,
                                  empty,
                                  the_spec_reqmts_trace,
                                  is_atomic => true,
                                  the_opr => the_operator);

              bind_operation ($3.psdl_id_value,
                              the_operator,
                              the_operation_map);
              the_spec_reqmts_trace := requirements(the_operator);
              the_spec_reqmts_trace := empty; }
        |
        ;

operator
        : OPERATOR_TOKEN IDENTIFIER
            { is_specification := true; }
          operator_spec
            { is_specification := false; }
          operator_impl
            { -- construct the psdl operator using the global variables
              build_psdl_operator($2.psdl_id_value,
                                  the_ada_name,
                                  the_gen_par,
                                  the_keywords,
                                  the_description,
                                  the_axioms,
                                  the_input,
                                  the_output,
                                  the_state,
```

```
                                    the_initial_expression_map,
                                    the_exceptions,
                                    the_specified_met,
                                    the_graph,
                                    the_streams,
                                    the_timers,
                                    the_trigger_map,
                                    the_exec_guard,
                                    the_out_guard,
                                    the_excep_trigger,
                                    the_timer_op,
                                    the_per,
                                    the_fw,
                                    the_mcp,
                                    the_mrt,
                                    the_impl_desc,
                                    the_reqmts_trace,
                                    is_atomic_operator,
                                    the_component);
              output_trace(the_component);
            }

;
operator_spec
        : SPECIFICATION_TOKEN
            { -- Initialize the variables used to build an operator
spec.
              the_gen_par := empty_type_declaration;
              the_input := empty_type_declaration;
              the_output := empty_type_declaration;
              the_state := empty_type_declaration;
              expression_sequence_pkg.empty(the_init_exp_seq);
              the_initial_expression_map := empty_init_map;
            the_exceptions := empty;
              the_specified_met := undefined_time; }
          interface
            { bind_initial_state(the_state, the_init_exp_seq,
                                 the_initial_expression_map); }
          functionality END_TOKEN
        ;

interface
        : interface attribute reqmts_trace
        |
        ;

attribute
        : GENERIC_TOKEN
            { the_type_decl := the_gen_par; }
          list_of_type_decl
            { the_gen_par := the_type_decl; }
        | INPUT_TOKEN
            { the_type_decl := the_input; }
          list_of_type_decl
            { the_input := the_type_decl; }
        | OUTPUT_TOKEN
            { the_type_decl := the_output; }
          list_of_type_decl
            { the_output := the_type_decl; }
        | STATES_TOKEN
```

```
            { the_states_token_line := current_line; -- For error
messages.
              the_states_token := convert(yytext); -- For error
messages.
              the_type_decl := the_state; }
          list_of_type_decl
            { the_state := the_type_decl; }
          INITIALLY_TOKEN initial_expression_list
            { expression_sequence_pkg.append(the_init_exp_seq,
                                             $6.expression_seq_value,
                                             the_init_exp_seq); }
        | EXCEPTIONS_TOKEN id_list
            { psdl_id_set_pkg.union(the_exceptions,
                                    to_set($2.psdl_id_sequence_value),
                                    the_exceptions ); }
        | MAXIMUM_TOKEN EXECUTION_TOKEN TIME_TOKEN time
            { the_specified_met := $4.integer_value; }
              -- Time is converted into millisec .
        ;

  -- Initialization of the_type_decl is done by the callers of this
rule.
list_of_type_decl
        : list_of_type_decl ',' type_decl
            { $$ := ( token_category => psdl_id_sequence_cat,
                      psdl_id_sequence_value =>

psdl_id_sequence_pkg.append($1.psdl_id_sequence_value,

$3.psdl_id_sequence_value )); }
        | type_decl
            { $$ := $1; }
        ;

type_decl
        : id_list ':' type_name
            { $$ := $1;
              bind_type_declaration($1.psdl_id_sequence_value,
                                    $3.type_name_value,
                              the_type_decl); }
        ;

type_name
        : IDENTIFIER
            { -- Save the previous value of the_type_decl.
            -- Needed because the list_of_type_decl below
            -- might contain nested type declarations.
            $$ := (token_category => type_decl_cat,
                      type_decl_value => the_type_decl);
              the_type_decl := empty_type_declaration; }
          '[' list_of_type_decl ']'
            { the_type_name :=
                create(name => $1.psdl_id_value,
                       formals => $4.psdl_id_sequence_value,
                       gen_par => the_type_decl);
            -- Now restore the previous value saved above.
            the_type_decl := $2.type_decl_value;
              $$ := (token_category => type_name_cat,
                      type_name_value => the_type_name); }
        | IDENTIFIER
            { the_type_name :=
```

```
                create(name => $1.psdl_id_value,
                       formals => psdl_id_sequence_pkg.empty,
                       gen_par => empty_type_declaration);
              $$ := (token_category =>  type_name_cat,
                     type_name_value =>  the_type_name); }
        ;

id_list
        : id_list ',' IDENTIFIER
            { $$ := ( token_category => psdl_id_sequence_cat,
                      psdl_id_sequence_value =>
                        add($3.psdl_id_value,
                            $1.psdl_id_sequence_value )); }
        | IDENTIFIER
            { $$ := ( token_category => psdl_id_sequence_cat,
                      psdl_id_sequence_value => add($1.psdl_id_value,
empty) ); }
        ;

        ------------------------------------------
        -- We use psdl_id_sequence as the data structure
        -- to store the requirement ids to include duplicates.
-- Psdl_id_set does not store duplicates.
        ------------------------------------------
reqmts_trace
        : REQ_BY_TOKEN  id_list
            { psdl_id_sequence_pkg.append(the_reqmts_trace,
                                          $2.psdl_id_sequence_value,
                                          the_reqmts_trace);
              psdl_id_sequence_pkg.append(the_spec_reqmts_trace,
                                          $2.psdl_id_sequence_value,
                                          the_spec_reqmts_trace); }
        |
        ;

functionality
        : keywords informal_desc formal_desc
        ;

keywords
        : KEYWORDS_TOKEN id_list
            { the_keywords := to_set($2.psdl_id_sequence_value); }
        |   { the_keywords := empty; }
        ;

informal_desc
        : DESCRIPTION_TOKEN TEXT_TOKEN
            { if is_specification then
                 the_description := $2.text_value;
              else the_impl_desc := $2.text_value;
              end if; }
        |
            { if is_specification then
                 the_description := empty;
              else the_impl_desc := empty;
              end if; }
        ;

formal_desc
        : axioms_TOKEN TEXT_TOKEN
            { the_axioms:= $2.text_value; }
```

```
        |
            { the_axioms:= empty; }
        ;

type_impl
        : IMPLEMENTATION_TOKEN ADA_TOKEN IDENTIFIER END_TOKEN
            { is_atomic_type := true;
              the_ada_name := ada_id($3.psdl_id_value); }
        | IMPLEMENTATION_TOKEN type_name op_impl_list END_TOKEN
            { is_atomic_type := false;
              the_data_structure := $2.type_name_value; }
        ;

op_impl_list
        : op_impl_list OPERATOR_TOKEN IDENTIFIER operator_impl
            { -- add implementation part to the operator in the
operation map
              add_op_impl_to_op_map($3.psdl_id_value,
                                    the_ada_name,
                                    is_atomic_operator,
                                    the_operation_map,
                                    the_graph,
                                    the_streams,
                                    the_timers,
                                    the_trigger_map,
                                    the_exec_guard,
                                    the_out_guard,
                                    the_excep_trigger,
                                    the_timer_op,
                                    the_per,
                                    the_fw,
                                    the_mcp,
                                    the_mrt,
                                    the_impl_desc ); }
        |
        ;

operator_impl
        : IMPLEMENTATION_TOKEN ADA_TOKEN IDENTIFIER END_TOKEN
            { is_atomic_operator := true;
              the_ada_name := ada_id($3.psdl_id_value); }
        | IMPLEMENTATION_TOKEN psdl_impl END_TOKEN
            { is_atomic_operator := false; }
        ;

psdl_impl
        : data_flow_diagram streams timers control_constraints
informal_desc
        ;

data_flow_diagram
        : { the_graph := empty_psdl_graph; }
          GRAPH_TOKEN vertex_list edge_list
        ;

                -- Time is the maximum execution time.
vertex_list
        : vertex_list VERTEX_TOKEN op_id optional_time graph_properties
            { the_graph := psdl_graph_pkg.add_vertex($3.op_id_value,
                                                     the_graph,
                                                     $4.integer_value,
```

```
$5.property_map_value); }
        |
        ;

                -- Time is the latency.
edge_list
        : edge_list EDGE_TOKEN IDENTIFIER
          optional_time op_id ARROW op_id graph_properties
            { the_graph := psdl_graph_pkg.add_edge($5.op_id_value,
                                                   $7.op_id_value,
                                                   $3.psdl_id_value,
                                                   the_graph,
                                                   $4.integer_value,

$8.property_map_value); }
        |
        ;

graph_properties
        : graph_properties PROPERTY_TOKEN IDENTIFIER '=' expression
            { bind($3.psdl_id_value, $5.expression_value,
$1.property_map_value);
              $$ := ( token_category => property_map_cat,
                      property_map_value => $1.property_map_value ); }
        |
            { $$ := ( token_category => property_map_cat,
                      property_map_value => empty_init_map ); }
        ;

op_id
        : operator_name opt_arg
            { $$ := ( token_category => op_id_cat,
                      op_id_value =>
                      ( type_name => $1.type_name_part,
                        operation_name => $1.op_name_part,
                        inputs => $2.input_value,
                        outputs => $2.output_value )); }
        ;

operator_name
        : IDENTIFIER '.' IDENTIFIER
            { $$ := ( token_category => operator_name_cat,
                      type_name_part => $1.psdl_id_value,
                      op_name_part => $3.psdl_id_value ); }
        | IDENTIFIER
            { $$ := ( token_category => operator_name_cat,
                      type_name_part => empty,
                      op_name_part => $1.psdl_id_value ); }
        ;

opt_arg
        : '(' optional_id_list '|' optional_id_list ')'
            { $$ := ( token_category => opt_arg_cat,
                      input_value => $2.psdl_id_sequence_value,
                      output_value => $4.psdl_id_sequence_value ); }
        |   { $$ := ( token_category => opt_arg_cat,
                      input_value => empty,
                      output_value => empty ); }
        ;
```

```
optional_id_list
        : id_list { $$ := $1; }
        | { $$ := ( token_category => psdl_id_sequence_cat,
                    psdl_id_sequence_value => empty ); }
        ;

optional_time
        : ':' time
            { $$ := (token_category => integer_cat,
                     integer_value => $2.integer_value); }
        |   { $$:= (token_category => integer_cat,
                    integer_value => undefined_time); }
        ;

streams
        : DATA_TOKEN STREAM_TOKEN
            { the_type_decl := empty_type_declaration; }
          list_of_type_decl
            { the_streams := the_type_decl; }
        |
            { the_streams := empty_type_declaration; }
        ;

        ------------------------------------------
        -- The order of id's is not important, so
        -- we use psdl_id_set as the data structure
        -- to store the timers.
        ------------------------------------------
timers
        : TIMER_TOKEN id_list
            { the_timers := to_set($2.psdl_id_sequence_value); }
        |   { the_timers := empty; }
        ;

control_constraints
        : CONTROL_TOKEN CONSTRAINTS_TOKEN
            { the_trigger_map := empty_trigger_map;
              the_per := empty_timing_map;
              the_fw := empty_timing_map;
              the_mcp := empty_timing_map;
              the_mrt := empty_timing_map;
              the_exec_guard := empty_exec_guard_map;
              the_out_guard := empty_out_guard_map;
              the_excep_trigger := empty_excep_trigger_map;
              the_timer_op := empty_timer_op_map; }
          constraints
        ;

constraints
        : constraints OPERATOR_TOKEN op_id
            { the_operator_id := $3.op_id_value;
              the_timer_op_set := timer_op_set_pkg.empty; }
          opt_trigger opt_period opt_finish_within
          opt_mcp opt_mrt constraint_options
        | OPERATOR_TOKEN op_id
            { the_operator_id := $2.op_id_value;
              the_timer_op_set := timer_op_set_pkg.empty; }
          opt_trigger opt_period opt_finish_within
          opt_mcp opt_mrt constraint_options
        ;
```

```
constraint_options
        : constraint_options OUTPUT_TOKEN
          id_list IF_TOKEN expression reqmts_trace
            { the_output_id.op := the_operator_id;
              for id: psdl_id in
                  psdl_id_sequence_pkg.scan($3.psdl_id_sequence_value)
              loop
                  the_output_id.stream := id;
                  bind(the_output_id, $6.expression_value,
the_out_guard);
              end loop; }
        | constraint_options EXCEPTION_TOKEN IDENTIFIER
          opt_if_predicate reqmts_trace
            { the_excep_id.op := the_operator_id;
              the_excep_id.excep := $3.psdl_id_value;
              bind(the_excep_id, $4.expression_value,
the_excep_trigger); }
        | constraint_options timer_op IDENTIFIER
          opt_if_predicate reqmts_trace
            { the_timer_op_record.op_id := $2.timer_op_id_value;
              the_timer_op_record.timer_id := $3.psdl_id_value;
              the_timer_op_record.guard := $5.expression_value;
              timer_op_set_pkg.add (the_timer_op_record,
                                    the_timer_op_set);
              bind(the_operator_id, the_timer_op_set, the_timer_op); }
        |
        ;

opt_trigger
        : TRIGGERED_TOKEN trigger opt_if_predicate reqmts_trace
          { bind(the_operator_id, $3.expression_value, the_exec_guard);
}
        |
        ;

trigger
        : BY_ALL_TOKEN id_list
            { the_trigger.tt := by_all;
              the_trigger.streams := to_set($2.psdl_id_sequence_value);
              bind(the_operator_id, the_trigger, the_trigger_map); }
        | BY_SOME_TOKEN id_list
            { the_trigger.tt := by_some;
              the_trigger.streams := to_set($2.psdl_id_sequence_value);
              bind(the_operator_id, the_trigger, the_trigger_map); }
        |   { the_trigger.tt := by_none;
              the_trigger.streams := empty;
              bind(the_operator_id, the_trigger, the_trigger_map); }
        ;


opt_period
        : PERIOD_TOKEN time reqmts_trace
            { bind(the_operator_id, $2.integer_value, the_per); }
        |
        ;

opt_finish_within
        : FINISH_TOKEN WITHIN_TOKEN time reqmts_trace
            { bind(the_operator_id, $3.integer_value, the_fw); }
        |
        ;
```

```
opt_mcp
        : MINIMUM_TOKEN CALL_PERIOD_TOKEN time reqmts_trace
          { bind(the_operator_id, $3.integer_value, the_mcp); }
        |
        ;

opt_mrt
        : max_resp_time time reqmts_trace
          { bind(the_operator_id, $2.integer_value, the_mrt); }
        |
        ;

max_resp_time
        : MAXIMUM_TOKEN RESPONSE_TOKEN TIME_TOKEN
        ;

timer_op
        : RESET_TOKEN
          { $$ := (token_category => timer_op_id_cat,
                   timer_op_id_value => t_reset); }
        | START_TOKEN
          { $$ := (token_category => timer_op_id_cat,
                   timer_op_id_value => t_start); }
        | STOP_TOKEN
          { $$ := (token_category => timer_op_id_cat,
                   timer_op_id_value => t_stop); }
        ;

opt_if_predicate
        : IF_TOKEN expression
          { $$ := (token_category => expression_cat,
                   expression_value => $2.expression_value); }
        | { $$ := (token_category => expression_cat,
                   expression_value => true_expression); }
        ;

       ------------------------------------------------------------------
--
       -- The expression sequence
       -- is used by procedure bind_initial_state together with
       -- the states map to construct the init_map.
       ------------------------------------------------------------------
--

initial_expression_list
        : initial_expression_list ',' initial_expression
          { $$ := (token_category =>  expression_seq_cat,
                   expression_seq_value =>
                     expression_sequence_pkg.add($3.expression_value,

$1.expression_seq_value )); }
        | initial_expression
          { $$ := (token_category => expression_seq_cat,
                   expression_seq_value =>
                     expression_sequence_pkg.add($1.expression_value,
                                                 empty_exp_seq )); }
        ;

       ------------------------------------------------------------------
--
```

```
        -- There is one and only one initial state(initial expression)
        -- for each state variable. This production returns one
        -- expression to the parent rule corresponding to one state.
        -- This is done by using the internal stack ($$ convention).
        ------------------------------------------------------------------
--

initial_expression
        : TRUE
            { $$ := (token_category => expression_cat,
                     expression_value => true_expression); }
        | FALSE
            { $$ := (token_category => expression_cat,
                     expression_value => false_expression); }
        | INTEGER_LITERAL
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_integer_literal($1.integer_value)); }
        | REAL_LITERAL
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_real_literal($1.text_value)); }
        | STRING_LITERAL
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_string_literal($1.text_value)); }
        | IDENTIFIER
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_identifier($1.psdl_id_value)); }
        | type_name '.' IDENTIFIER
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_function_call($1.type_name_value,
                                          psdl_id($3.psdl_id_value),
                                          empty_exp_seq)); }
        | type_name '.' IDENTIFIER '(' initial_expression_list ')'
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_function_call($1.type_name_value,
                                          psdl_id($3.psdl_id_value),
                                          $5.expression_seq_value));}
        | '(' initial_expression ')'
            { $$ := (token_category => expression_cat,
                     expression_value => $2.expression_value) ;
                       }
        | initial_expression log_op initial_expression   %prec
logical_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value));
 }
        | initial_expression rel_op initial_expression %prec
relational_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value));
```

```
 }
        | '-' initial_expression                %prec
unary_adding_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("-"),
                                       $2.expression_value )); }
        | '+' initial_expression                %prec
unary_adding_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("+"),
                                       $2.expression_value ));}
        | initial_expression bin_add_op initial_expression
          %prec binary_adding_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value)); }
        | initial_expression bin_mul_op initial_expression
          %prec multiplying_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value)); }
        | initial_expression EXP_TOKEN initial_expression
          %prec highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       convert("**"),
                                       $3.expression_value));
 }
        | NOT_TOKEN initial_expression         %prec
highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("NOT"),
                                       $2.expression_value )); }
        | ABS_TOKEN initial_expression         %prec
highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("ABS"),
                                       $2.expression_value )); }
        ;

log_op
        : AND_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("AND") ); }
        | OR_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("OR") ); }
        | XOR_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("XOR") ); }
        ;
```

```
rel_op
        : '<'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("<") ); }
        | '>'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert(">") ); }
        | '='
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("=") ); }
        | GREATER_THAN_OR_EQUAL
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert(">=") ); }
        | LESS_THAN_OR_EQUAL
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("<=") ); }
        | INEQUALITY
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("/=") ); }
        ;

bin_add_op
        : '+'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("+") ); }
        | '-'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("-") ); }
        | '&'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("&") ); }
        ;

bin_mul_op
        : '*'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("*") ); }
        | '/'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("/") ); }
        | MOD_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("MOD") ); }
        | REM_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("REM") ); }
        ;


time
        : time_number MICROSEC_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => ($1.integer_value + 999)/1000); }
        | time_number MS_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value); }
        | time_number SEC_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value * 1000); }
        | time_number MIN_TOKEN
            { $$ := (token_category => integer_cat,
```

```
                integer_value => $1.integer_value * 60000); }
        | time_number HOURS_TOKEN
            { $$ := (token_category => integer_cat,
                integer_value => $1.integer_value * 3600000); }
        ;

time_number
        : INTEGER_LITERAL
            { $$ := (token_category => integer_cat,
                integer_value => $1.integer_value); }
        ;


expression_list
        : expression_list ',' expression
            { $$ := (token_category =>  expression_seq_cat,
                expression_seq_value =>
                expression_sequence_pkg.add($3.expression_value,
                                            $1.expression_seq_value
)); }
        | expression
            { $$ := (token_category =>  expression_seq_cat,
                expression_seq_value =>
                expression_sequence_pkg.add($1.expression_value,
                                            empty_exp_seq )); }
        ;

-------------------------------------------------------------------------
-- Expressions can appear in guards appearing in control constraints.
-- These guards can be associated with triggering conditions, or
-- conditional outputs, conditional exceptions, or conditional timer
-- operations. Similar to initial expression, except that tim e values
-- and references to timers and data streams are allowed.
-------------------------------------------------------------------------
expression
        : TRUE
            { $$ := (token_category => expression_cat,
                expression_value => true_expression); }
        | FALSE
            { $$ := (token_category => expression_cat,
                expression_value => false_expression); }
        | INTEGER_LITERAL
            { $$ := (token_category => expression_cat,
                expression_value =>
                create_integer_literal($1.integer_value)); }
        | REAL_LITERAL
            { $$ := (token_category => expression_cat,
                expression_value =>
                create_real_literal($1.text_value)); }
        | STRING_LITERAL
            { $$ := (token_category => expression_cat,
                expression_value =>
                create_string_literal($1.text_value)); }
        | IDENTIFIER
            { $$ := (token_category => expression_cat,
                expression_value =>
                create_identifier($1.psdl_id_value)); }
         -- The only difference from the initial expression
        | time
            { $$ := (token_category => expression_cat,
                expression_value =>
```

```
    create_time_literal (natural($1.integer_value)));}
| type_name '.' IDENTIFIER
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_function_call($1.type_name_value,
                                  psdl_id($3.psdl_id_value),
                                  empty_exp_seq)); }
| type_name '.' IDENTIFIER '(' expression_list ')'
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_function_call($1.type_name_value,
                                  psdl_id($3.psdl_id_value),
                                  $5.expression_seq_value));}
| '(' expression ')'
    { $$ := (token_category => expression_cat,
             expression_value => $2.expression_value);
              }
| expression log_op expression   %prec logical_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               $2.psdl_id_value,
                               $3.expression_value)); }
| expression rel_op expression %prec relational_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               $2.psdl_id_value,
                               $3.expression_value)); }
| '-' expression                %prec unary_adding_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_unary_op  (convert("-"),
                               $2.expression_value)); }
| '+' expression                 %prec unary_adding_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_unary_op  (convert("+"),
                               $2.expression_value)); }
| expression bin_add_op expression
  %prec binary_adding_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               $2.psdl_id_value,
                               $3.expression_value)); }
| expression bin_mul_op expression
  %prec multiplying_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               $2.psdl_id_value,
                               $3.expression_value)); }
| expression EXP_TOKEN expression
  %prec highest_precedence_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               convert("**"),
                               $3.expression_value)); }
| NOT_TOKEN expression          %prec highest_precedence_operator
```

```
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_unary_op  (convert("NOT"),
                                       $2.expression_value)); }
        | ABS_TOKEN expression          %prec highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_unary_op  (convert("ABS"),
                                       $2.expression_value)); }
        ;


%%

----------=================================================----------
--
--                          package spec parser
--
----------=================================================----------
  with psdl_program_pkg; use psdl_program_pkg;
  with text_io; use text_io;
package parser is
  procedure get(item: in out psdl_program);
  procedure get(file: in file_type; item: in out psdl_program);

  semantic_error: exception;
end parser;

----------=============================================================----------
--
--                          package body  parser
--
----------=============================================================----------

  with parser_tokens; use parser_tokens;
  with parser_goto; use parser_goto;
  with parser_shift_reduce; use parser_shift_reduce;
  with parser_lex; use parser_lex;
  with parser_lex_dfa; use parser_lex_dfa;
  with psdl_component_pkg; use psdl_component_pkg;
  with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
  with psdl_graph_pkg; use psdl_graph_pkg;
  with psdl_io; use psdl_io;
  with expression_pkg; use expression_pkg;
package body parser is
  subtype exp_seq is expression_sequence_pkg.sequence;

  function empty_exp_seq return expression_sequence
    renames expression_sequence_pkg.empty;
    -- Returns an empty expression sequence.

  -- global variables used by the parser.
  the_program: psdl_program;
  the_component: psdl_component;
  the_operator: operator;
  the_atomic_type: atomic_type;
  the_atomic_operator: atomic_operator;
  the_composite_type: composite_type;
  the_composite_operator: composite_operator;
  the_ada_name: ada_id;
  the_gen_par: type_declaration;
```

```
    the_type_gen_par: type_declaration;
    the_keywords: psdl_id_set;
    the_reqmts_trace: psdl_id_sequence;
    the_spec_reqmts_trace: psdl_id_sequence;
    the_description: text;
    the_axioms: text;
    the_output_id: output_id;
      -- a temporary variable to hold output_id to construct out_guard map
    the_excep_id: excep_id;
      -- a temporary variable to hold excep_id to construct excep_trigger
map
    the_model: type_declaration;
    the_operation_map: operation_map;
    the_data_structure: type_name;
    the_input: type_declaration;
    the_output: type_declaration;
    the_state: type_declaration;
    the_states_token_line: natural;
    the_states_token: text;
    the_initial_expression_map: init_map;
    the_exceptions: psdl_id_set;
    the_specified_met: millisec;
    the_graph: psdl_graph;
    the_streams: type_declaration;
    the_timers: psdl_id_set;
    the_trigger_map: trigger_map;
    the_exec_guard: exec_guard_map;
    the_out_guard: out_guard_map;
    the_excep_trigger: excep_trigger_map;
    the_timer_op: timer_op_map;
    the_per: timing_map;
    the_fw: timing_map;
    the_mcp: timing_map;
    the_mrt: timing_map;
    the_operator_id: op_id;
      -- is used for storing the operator id's in control constraints part
    is_atomic_type: boolean;
      -- true if the psdl_component is an atomic type.
    is_atomic_operator: boolean;
      -- true if the psdl_component is an atomic operator.
    is_specification: boolean;
     -- True if the current unit is a psdl specification part.
    the_init_exp_seq: exp_seq;
      -- Holds the initial expressions for all state variables in an
operator spec.
    the_type_name: type_name;
    the_type_decl: type_declaration;
      -- Used to hold an inherited/synthesized attribtue pair.
    the_trigger: trigger;
    the_timer_op_record: timer_op;
    the_timer_op_set: timer_op_set;
    the_impl_desc: text;

    ------------------------------------------------------------------
    -- procedure initialize_state_variables
    ------------------------------------------------------------------
    procedure initialize_state_variables is
    begin
      yyval := (token_category => no_value_cat);
    end initialize_state_variables;
```

```
    ------------------------------------------------------------
    -- procedure yyparse
    ------------------------------------------------------------
    procedure yyparse; -- Body is automatically generated.

    ------------------------------------------------------------
    -- procedure yyerror
    ------------------------------------------------------------
    procedure yyerror(s: in string := "syntax error";
                      err_line: natural := current_line;
                      err_token: text := convert(yytext)) is
      space: integer;
    begin
      new_line;
      put_line(standard_error,
               "line" & integer'image(err_line) & ": " &
 convert(err_token));
      space := integer'image(err_line)'length +
 integer(convert(err_token)'length) + 5;
      for i in 1 .. space loop put(standard_error, "-"); end loop;
      put_line(standard_error, "^ " & s);
    end yyerror;

    ------------------------------------------------------------
    -- given a string of characters corresponding to a natural number,
    -- returns the natural value
    ------------------------------------------------------------
    function convert_to_natural(string_digit: string) return natural is
      digit, value : natural := 0;
    begin
      for i in 1 ..  string_digit'length loop
        case string_digit(i) is
          when '0' => digit := 0;
          when '1' => digit := 1;
          when '2' => digit := 2;
          when '3' => digit := 3;
          when '4' => digit := 4;
          when '5' => digit := 5;
          when '6' => digit := 6;
          when '7' => digit := 7;
          when '8' => digit := 8;
          when '9' => digit := 9;
          when others => return value;
        end case;
        value := (10 * value) + digit;
      end loop;
      return value;
    end convert_to_natural;

    ------------------------------------------------------------
    --                        procedure get
    --
    --  reads the standard input, parses it, and creates the
    --  psdl adt.
    ------------------------------------------------------------
procedure get(item: in out psdl_program) is
begin
  initialize_state_variables;
  initialize_yylex;
  yyparse;
  assign(item, the_program);
```

```
end get;

  ------------------------------------------------------------------
  --                          procedure get

  --  reads the psdl source file from a file,
  --  parses it, and creates the psdl adt.
  ------------------------------------------------------------------
  procedure get(file: in file_type;
                item: in out psdl_program ) is
  begin
    set_input(file);
    get(item);
    set_input(standard_input);
  end get;

  ------------------------------------------------------------------
  --                      procedure bind_type_declaration
  --
  --bind each id in id the id
  --set to the type name
  --return temp_type_decl
  ------------------------------------------------------------------
  procedure bind_type_declaration(i_s: in psdl_id_sequence; tn: in
type_name;
                                  td: in out type_declaration) is
  begin
     for id : psdl_id in psdl_id_sequence_pkg.scan(i_s) loop
       bind(id, tn, td);
     end loop;
  end bind_type_declaration;

  ------------------------------------------------------------------
  --                      procedure bind_initial_state
  --
  -- Bind each id in the state map domain
  -- set to the type name initial expression
  ------------------------------------------------------------------
  procedure bind_initial_state(state: in type_declaration; init_seq: in
exp_seq;
                               init_exp_map: in out init_map) is
    i: natural := 0;
  begin
-- Added by Dave Dampier 20 April 1994 to eliminate use of the M4
Macros,
-- and adopt use of the new generator processor for loops.

-- Begin syntax for generate expansion.
    for id : psdl_id, td : type_name in type_declaration_pkg.scan(state)
loop
      i := i + 1;
      if i > expression_sequence_pkg.length(init_seq) then
         yyerror("semantic error - some states are not initialized.",
                  the_states_token_line, the_states_token);
         raise semantic_error;
      else bind(id,
                expression_sequence_pkg.fetch(init_seq, i),
                init_exp_map);
      end if;
    end loop;
-- End of Added Code.
```

```
-- Also eliminated old M4 code.

    if i < expression_sequence_pkg.length(init_seq) then
       yyerror("semantic error - there are more initializations than the
states",
               the_states_token_line, the_states_token);
       raise semantic_error;
    end if;
  end bind_initial_state;

  ------------------------------------------------------------------
  --                        procedure make_psdl_type
  --
  --   construct the  psdl type using global variables
  --
  ------------------------------------------------------------------
  procedure build_psdl_type(c_name: in psdl_id;
                            c_a_name: in ada_id;
                            mdl: in type_declaration;
                            d_str: in type_name;
                            ops: in operation_map;
                            g_par: in type_declaration;
                            kwr: in psdl_id_set;
                            i_desc: in text;
                            f_desc: in text;
                            is_atomic: in boolean;
                            the_type: in out data_type;
                            reqms: in psdl_id_sequence) is
  begin
    if is_atomic then
      the_type := make_atomic_type(psdl_name => c_name,
                                   ada_name => c_a_name,
                                   model => mdl,
                                   gen_par => g_par,
                                   operations=> ops,
                                   keywords => kwr,
                                   informal_description => i_desc,
                                   axioms => f_desc,
                                   requirements => reqms );
    else the_type := make_composite_type(name => c_name,
                                         model => mdl,
                                         data_structure => d_str,
                                         operations=> ops,
                                         gen_par => g_par,
                                         keywords => kwr,
                                         informal_description => i_desc,
                                         axioms => f_desc,
                                         requirements => reqms );
    end if;
  end build_psdl_type;

  ------------------------------------------------------------------
  --   construct the  psdl operator using global variables
  ------------------------------------------------------------------
  procedure build_psdl_operator(c_name: in psdl_id;
                                c_a_name: in ada_id;
                                g_par: in type_declaration;
                                kwr: in psdl_id_set;
                                i_desc: in text;
                                f_desc: in text;
                                inp: in type_declaration;
```

```
                                    otp: in type_declaration;
                                    st: in type_declaration;
                                    i_exp_map: in init_map;
                                    excps: in psdl_id_set;
                                    s_met: in millisec;
                                    gr: in psdl_graph;
                                    d_stream: in type_declaration;
                                    tmrs: in psdl_id_set;
                                    trigs: in trigger_map;
                                    e_guard: in exec_guard_map;
                                    o_guard: in out_guard_map;
                                    e_trigger: in excep_trigger_map;
                                    t_op: in timer_op_map;
                                    per: in timing_map;
                                    fw: in timing_map;
                                    mcp: in timing_map;
                                    mrt: in timing_map;
                                    im_desc: in text;
                                    reqms: in psdl_id_sequence;
                                    is_atomic: in boolean;
                                    the_opr: in out operator ) is
  begin
    if is_atomic then
       the_opr := make_atomic_operator(psdl_name => c_name,
                                       ada_name => c_a_name,
                                       gen_par => g_par,
                                       keywords => kwr,
                                       informal_description => i_desc,
                                       axioms => f_desc,
                                       input => inp,
                                       output => otp,
                                       state => st,
                                       initialization_map => i_exp_map,
                                       exceptions => excps,
                                       specified_met => s_met,
                                       requirements => reqms);
    else the_opr := make_composite_operator(name => c_name,
                                            gen_par => g_par,
                                            keywords => kwr,
                                            informal_description =>
i_desc,
                                            axioms => f_desc,
                                            input => inp,
                                            output => otp,
                                            state => st,
                                            initialization_map =>
i_exp_map,
                                            exceptions => excps,
                                            specified_met => s_met,
                                            graph => gr,
                                            streams => d_stream,
                                            timers => tmrs,
                                            trigger => trigs,
                                            exec_guard=> e_guard,
                                            out_guard => o_guard,
                                            excep_trigger => e_trigger,
                                            timer_op => t_op,
                                            per => per,
                                            fw => fw,
                                            mcp => mcp,
                                            mrt => mrt,
```

```
                                                impl_desc => im_desc,
                                                requirements => reqms);
      end if;
    end build_psdl_operator;

    ------------------------------------------------------------------
    --                 procedure add_op_impl_to_op_map
    --
    --   Augments the operation map we constructed using only the
    --   specification part.
    --   Fetches the operator from the map, with the information
    --   from the specification part and adds the implementation to it.
    --   Removes the old one, and adds the new complete operator the map.
    ------------------------------------------------------------------
    procedure add_op_impl_to_op_map(op_name: in psdl_id;
                                    a_name: in ada_id;
                                    is_atomic: in boolean;
                                    o_map: in out operation_map;
                                    gr: in out psdl_graph;
                                    d_stream: in out type_declaration;
                                    tmrs: in out psdl_id_set;
                                    trigs: in out trigger_map;
                                    e_guard: in out exec_guard_map;
                                    o_guard: in out out_guard_map;
                                    e_trigger: in out excep_trigger_map;
                                    t_op: in out timer_op_map;
                                    per: in out timing_map;
                                    fw: in out timing_map;
                                    mcp: in out timing_map;
                                    mrt: in out timing_map;
                                    im_desc: in out text ) is
      temp_op: operator;
    begin
      if operation_map_pkg.member(op_name, operation_map_pkg.map(o_map))
then
         temp_op := operation_map_pkg.fetch(operation_map_pkg.map(o_map),
op_name);
         operation_map_pkg.remove(op_name, operation_map_pkg.map(o_map));
         if is_atomic then
            temp_op := make_atomic_operator
                               (psdl_name => op_name,
                                ada_name => a_name,
                                gen_par => generic_parameters(temp_op),
                                keywords => keywords(temp_op),
                                informal_description =>
informal_description(temp_op),
                                axioms => axioms(temp_op),
                                input => inputs(temp_op),
                                output => outputs(temp_op),
                                state => states(temp_op),
                                initialization_map =>
get_init_map(temp_op),
                                exceptions=> exceptions(temp_op),
                                specified_met =>
specified_maximum_execution_time(temp_op) );
         else
            temp_op := make_composite_operator
                               (name => op_name,
                                gen_par => generic_parameters(temp_op),
                                keywords => keywords(temp_op),
```

```
                                informal_description =>
informal_description(temp_op),
                                axioms => axioms(temp_op),
                                input => inputs(temp_op),
                                output => outputs(temp_op),
                                state => states(temp_op),
                                initialization_map =>
get_init_map(temp_op),
                                exceptions=> exceptions(temp_op),
                                specified_met =>

specified_maximum_execution_time(temp_op),
                                graph => gr,
                                streams => d_stream,
                                timers => tmrs,
                                trigger => trigs,
                                exec_guard=> e_guard,
                                out_guard => o_guard,
                                excep_trigger => e_trigger,
                                timer_op => t_op,
                                per => per,
                                fw => fw,
                                mcp => mcp,
                                mrt => mrt,
                                impl_desc => im_desc);
       end if;
       bind_operation(op_name, temp_op, o_map);
       -- reset everything after you are done.(the variables that have
default values)
       gr := empty_psdl_graph;
       d_stream := empty_type_declaration;
       tmrs := empty;
       trigs := empty_trigger_map;
       e_guard := empty_exec_guard_map;
       o_guard := empty_out_guard_map;
       e_trigger := empty_excep_trigger_map;
       t_op := empty_timer_op_map;
       per := empty_timing_map;
       fw := empty_timing_map;
       mcp := empty_timing_map;
       mrt := empty_timing_map;
       im_desc := empty;
    else
      put(standard_error, "warning: the specification of operator '");
      put_line(standard_error,
               convert(op_name) & "' was not given, implementation
ignored.");
    end if;
  end add_op_impl_to_op_map;


  ------------------------------------------------------------------
  --                     procedure output_trace
  --
  --pass the pointer to the component to retrieve the set of
  --requirements ids. Call put_reqm_ids defined in psdl_io package
  --to output component name and ids only if there are any ids.
  ------------------------------------------------------------------
  procedure output_trace(the_type: psdl_component) is

     reqts : psdl_id_sequence;
```

```
   begin
       reqts := the_reqmts_trace;
--     reqts := requirements(the_type);  For testing purposes
      if not equal (reqts,empty) then
        put_reqm_ids(the_type, reqts);
      end if;
     the_reqmts_trace := empty;
     the_spec_reqmts_trace := empty;
   end output_trace;


  -- Generated body of yyparse goes here.
##%procedure_parse

end parser;
```

LIST OF REFERENCES

[1] J. Self. Ayacc User's Manual. Arcadia Environment Research Project. Version 1.0, May 1988.

[2] Luqi, V. Berzins, and R. T. Yeh. A Prototyping Language for Real-Time Software. *IEEE Transactions on Software Engineering*, October 1988.

[3] Luqi. Real-Time Constraints In A Rapid Prototyping Language. *Computer Languages*, 1993.

[4] V. Berzins and Luqi, B. Kraemer. Compositional semantics of a real-time prototyping language. *IEEE Transactions on Software Engineering*, 1992.

[5] V. Berzins and Luqi. Semantics of a real-time language in Proc. Real-Time Syst. Symp., Huntsville, AL., Dec. 1988, Computer Society Press.

[6] Bindu Rao. Object-Oriented Databases; Technology, Applications, and Products.

[7] J. Self. Aflex – An Ada Lexical Analyzer Generator. Arcadia Environment Research Project. Version 1.0, May 1990.

[8] Luqi and V. Berzins. Rapid Prototyping of Real-Time Systems. *IEEE Software*, September 1988.

[9] M. Card. FIRM: An Ada Binding to ODMG-93 1.2, STC 96.

[10] D. Barry. "Charting the Feature Coverage of ODBMSs", *Object Magazine*, February 1997.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Prof. Valdis Berzins 1
   Naval Postgraduate School
   Code CS/Be
   Monterey, California 93943-5101

4. Dave J. Schmidt 1
   808 Sutter Street
   San Diego, California 92013

5. John Schmidt 1
   406 East Third Street
   Wahoo, Nebraska 68066